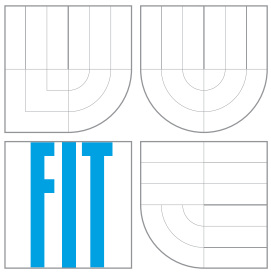
BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

GNOME KEYRING STORAGE IN FREEIPA

VYUŽITÍ KLÍČENKY GNOME V PROJEKTU FREEIPA

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
Bc. MICHAL ŽIDEK

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Židek Michal, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Využití klíčenky Gnome v projektu FreeIPA**
GNOME Keyring Storage in FreeIPA
Kategorie: Bezpečnost

Pokyny:

1. Nastudujte projekt FreeIPA. Zaměřte se na správu dat týkající se identity uživatelů. Nastudujte část Vault projektu FreeIPA. Zaměřte se na rozhraní JSON-RPC této části. Nastudujte projekt Klíčenka Gnome.
2. Analyzujte práci s daty v klíčence Gnome. Navrhněte modul pro klíčenku Gnome, který zprostředkujte část Vault projektu FreeIPA pro účely alternativního úložiště citlivých dat.
3. Navrhněte nutné úpravy projektu FreeIPA a klíčenky Gnome, příp. souvisejících komponent pro integraci navrženého modulu. Implementujte prototyp vašeho řešení.
4. Ověřte správnost vašeho řešení pomocí automatizovaných testů na různých konfiguracích FreeIPA a klíčenky Gnome.

Literatura:

1. Domovská stránka projektu FreeIPA. URL: https://www.freeipa.org/
2. Domovská stránka projektu Klíčenka Gnome. URL: https://wiki.gnome.org/Projects/GnomeKeyring

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D.**, UITS FIT VUT
Konzultant: Košek Martin, Ing., RHcz
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstract

This master's thesis gives introduction to FreeIPA project and GNOME Keyring project. It discusses benefits of possible integration of GNOME Keyring into FreeIPA using FreeIPA's component called Password Vault. Designs of possible implementations are provided. Prototype of one of these designs is implemented.

Abstrakt

Tato diplomová práce poskytuje úvod do projektu FreeIPA a projektu GNOME Keyring. Rozebírá možné výhody integrace GNOME Keyringu do FreeIPA pomocí komponenty FreeIPA zvané Password Vault. Jsou poskytnuty návrhy možných implementací a rozebírají se jejich výhody. Jeden z návrhů je pak implementován ve formě prototypu.

Keywords

GNOME, GNOME Keyring, FreeIPA, Password Vault, SSSD, keyring, secure storage, desktop environment.

Klíčová slova

GNOME, GNOME Keyring, FreeIPA, Password Vault, SSSD, klíčenka, bezpečné úložiště, desktopové prostředí.

Reference

ŽIDEK, Michal. *GNOME Keyring Storage in FreeIPA*. Brno, 2016. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Smrčka Aleš.

GNOME Keyring Storage in FreeIPA

Declaration

Hereby I declare that this masters's thesis was prepared as an original author's work under the supervision of Ing. Aleš Smrčka, Ph.D. The supplementary information was provided by Ing. Martin Košek and Alexander Bokovoy. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Michal Židek
May 25, 2016

Acknowledgements

Here, I would like to thank my supervisor Ing. Aleš Smrčka, Ph.D., Ing. Martin Košek and Alexander Bokovoy for their suggestions, guidance and help during my work on this thesis.

Contents

Chapter 1

Introduction

FreeIPA[1] is an identity management system that consists of multiple components each providing some service for administrators of domains and their users. One of FreeIPA components provides service called Password Vault. This service provides safe storage for secrets. Users can use this service from any computer enrolled in the FreeIPA domain. GNOME Keyring project also provides safe storage for secrets, however, does not operate across multiple computers in a network environment. GNOME Keyring and it's API is used by many applications and some of them may benefit if GNOME Keyring was integrated with FreeIPA allowing them to share secrets from multiple computers on one place while still using the same API. This master's theses investigates how this integration could be implemented, describes implementation of one prototype solution and discusses it's advantages and shortcomings.

Chapters 2 and 3 provide basic information about project FreeIPA and it's components, with focus on Password Vault. In Chapter 4, basic information about project GNOME Keyring is provided. These chapters aim to give the reader theoretical background for discussion about possible integration of these two projects. Chapter 5 discusses multiple ways how the actual implementation could be done and explains the choice of approach used in the prototype and describes it in more detail. An automated set of tests was created to test functionality of this prototype. Chapter 6 explains how the prototype was tested with automated tests. Some notes about current upstream efforts related to this topic as well summary of the work being done as part of this thesis are provided in Chapter 7.

[1] IPA stands for Identity Policy Audit

Chapter 2

Introduction to FreeIPA

This chapter provides basic information about the FreeIPA project. Section 2.1 explains the term identity management. Section 2.2 will provide details about server side components of FreeIPA. Section 2.3 details how the users and administrators interact with FreeIPA. Section 2.4 shows how machines can join the FreeIPA domain. Section 2.5 provides details about how other programs can communicate with it and Section 2.6 introduces a component called Password Vault.

2.1 Identity Management

FreeIPA is an open-source **identity management solution** or **identity management system** (IdM for short) for Linux/UNIX network environment [7]. IdM is a system to manage identities, their authentication mechanisms, authorization policies and other related tasks in a domain or across several domains. The identities are not only users, but also groups, services or hosts. IdM systems aim to simplify and reduce repetitive tasks associated with managing identities and as such are necessary components for larger companies, schools, service providers or other institutions that need to manage large number of identities.

2.2 FreeIPA Server Components

FreeIPA project brings several components that provide functionality commonly needed in identity management systems under one roof and aims to empower administrators with simple and unified way for installation, configuration and management of these components. However FreeIPA should not be seen as only shelter for other existing components, because the project brings a lot of new additional features and components that are specific to FreeIPA installations.

The official website [7] lists some of the components in the description of FreeIPA project, which is the following: *„FreeIPA is an integrated security information management solution combining* **Linux** *(Fedora),* **389 Directory Server** [1]*,* **MIT Kerberos**[2]*,* **NTP**[3]*,* **DNS**[4]*,*

[1] 389 Directory Server is an implementation of LDAP server. See http://www.port389.org/
[2] Kerberos is a network authentication protocol. See http://web.mit.edu/kerberos/
[3] Network Time Protocol. See http://www.ntp.org/
[4] FreeIPA has integrated DNS server. See http://www.freeipa.org/page/DNS for more details.

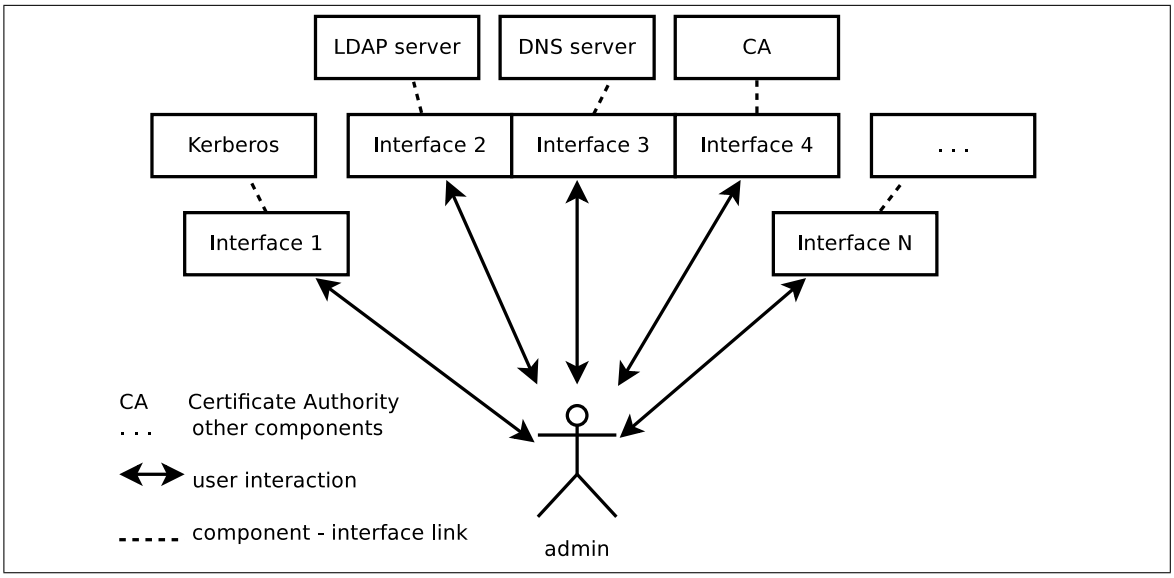


Figure 2.1: Managing components without FreeIPA.

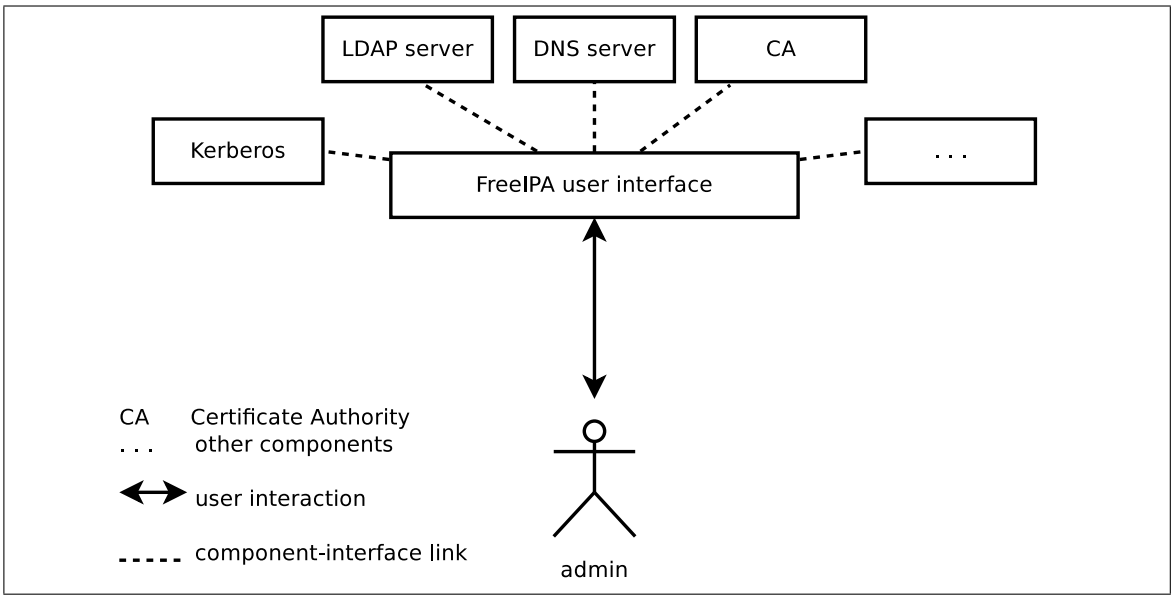


Figure 2.2: Simplified management with FreeIPA.

Dogtag *(Certificate System)*[5]. *It consists of a web interface and command-line administration tools. It is an integrated Identity and Authentication solution for Linux/UNIX networked environments. A FreeIPA server provides centralized authentication, authorization and account information by storing data about users, groups, hosts and other objects necessary to manage the security aspects of a network of computers.“ [7]*

We can look at these components separately and use their respective tools to manage them as if they were not part of FreeIPA server. For example the directory server can be queried and modified using the traditional LDAP tools such as `ldapsearch`, `ldapadd` and `ldapmodify`. While for querying the database, this is safe and can be used if administrators

[5]The Dogtag Certificate System is an open source Certificate Authority. See http://pki.fedoraproject.org/wiki/PKI_Main_Page

prefer them, modifying the database directly using these tools is more error prone, so it is better to use tools provided by FreeIPA if possible, to avoid inconsistencies in the database. We will talk more about FreeIPA's user interface later in this chapter. Figure **2.3** shows how `ipa`[6] tool is used to search information about a user and Figure **2.4** shows the same user found with `ldapsearch`.

```
$ ipa user-find --raw admin
--------------
1 user matched
--------------
  uid: admin
  sn: Administrator
  homedirectory: /home/admin
  loginshell: /bin/bash
  uidnumber: 1514600000
  gidnumber: 1514600000
  nsaccountlock: FALSE
  has_password: TRUE
  has_keytab: TRUE
----------------------------
Number of entries returned 1
----------------------------
```

Figure 2.3: An example of FreeIPA server query using the `ipa` tool.

The upstream development team focuses primarily on bringing FreeIPA server to Fedora, Red Hat Enterprise Linux and CentOS GNU/Linux distributions, however there are efforts to package FreeIPA server to some other distributions [7].

2.2.1 Installing FreeIPA Server Components

To simplify installation of FreeIPA server components, a tool written in Python called `ipa-server-install` is provided. This tool can be used in an interactive way in which it asks user to type in all necessary options step by step or alternatively this information can be passed through command line arguments. The tool makes a good job informing the user about each step it makes, so reading it's output will give us good overview of what components FreeIPA uses on server side. Depending on what optional components are installed on the FreeIPA server the list of installed components could be following:

- NTP deamon (ntpd),
- Diretory server (dirsrv),
- Certificate server (pki-tomcatd),
- Kerberos KDC (krb5kdc),
- kadmin,

[6] Lowercase `ipa` refers to the command line tool.
[7] Ubuntu packaging efforts are tracked on this site https://launchpad.net/freeipa

- ipa_memcached,
- ipa-otpd,
- web interface (httpd),
- DNS (named), and
- DNS key synchronization service (ipa-dnskeysyncd).

After successful installation of server components, the **ipactl**[8] tool can be used to show status of these components as shown on Figure 2.2.1. This tool is also used for starting, stopping or restarting FreeIPA components.

2.3 User Interface of FreeIPA Server

FreeIPA provides both command line interface as well as web browser interface. They both aim to provide full access to FreeIPA management capabilities, so that users can choose the interface that suites them more. However there are some features or plugin provided functionality that is not yet accessible through both interfaces.

[8] See **man 8 ipactl**

```
$ ldapsearch -x -h ipa.example.test \
> -b uid=admin,cn=users,cn=accounts,dc=example,dc=test
<snip>
# admin, users, accounts, example.test
dn: uid=admin,cn=users,cn=accounts,dc=example,dc=test
objectClass: top
objectClass: person
objectClass: posixaccount
objectClass: krbprincipalaux
objectClass: krbticketpolicyaux
objectClass: inetuser
objectClass: ipaobject
objectClass: ipasshuser
objectClass: ipaSshGroupOfPubKeys
uid: admin
cn: Administrator
sn: Administrator
uidNumber: 1686400000
gidNumber: 1686400000
homeDirectory: /home/admin
loginShell: /bin/bash
gecos: Administrator
<snip>
```

Figure 2.4: Example of `ldapsearch` query for FreeIPA in ipa.example.test domain (sanitized output).

```
# ipactl status
Directory Service: RUNNING
krb5kdc Service: RUNNING
kadmin Service: RUNNING
named Service: RUNNING
ipa_memcached Service: RUNNING
httpd Service: RUNNING
pki-tomcatd Service: RUNNING
ipa-otpd Service: RUNNING
ipa-dnskeysyncd Service: RUNNING
ipa: INFO: The ipactl command was successful
```

Figure 2.5: Using ipactl to show status of FreeIPA components.

2.3.1 Command Line Interface

The main command line interface for FreeIPA interaction is the `ipa` tool. It is a Python script and it's simplified usage[9] is following:

```
ipa COMMAND [parameters]
```

This script was briefly shown already on Figure **2.3** where the `COMMAND` was `user-find` with parameter `admin`. The `user-find` command is a built-in ipa command. **Built-in commands** are commands that are available on all FreeIPA installations and they provide core functionality. Second category of commands that ipa tool provides are **plugin provided commands**. As the name suggests, these commands are provided by plugins that can be installed additionally, thus not all FreeIPA installations will have them [15].

The naming of all commands follows a certain pattern which is used by both built-in and plugin provided commands to allow smoother experience with this tool. The pattern is *topic-action*, for example in the example on Figure **2.3**, the topic is *user* and the action is *find*. So the command is `user-find`. The list of all available topics can be retrieved by typing `ipa help topics` in the terminal as shown on the figure 2.6. To get help on particular topic, the `ipa help topic_name` pattern is used, which will usually provide short introduction to the topic as well as list of available commands in that topic. Also all commands accept `--help` as a parameter which prints that commands help.

In order to use the ipa tool, user must have valid Kerberos ticket that will authenticate him against FreeIPA. Many of the ipa commands can be executed only by FreeIPA administrator.

2.3.2 Web Browser Interface

Along with command line interface FreeIPA also provides web user interface for people who prefer graphical user interfaces. FreeIPA server is providing this interface on standard HTTPS port so simply going to the FreeIPA server URL in web browser will take the users to the login screen where they can authenticate with their user name and password as shown on figure 2.7. The login page will also provide users with link to page where they can learn how to enable Kerberos authentication for this interface in their web browsers. With Kerberos authentication, users only need to have valid Kerberos ticket to access FreeIPA's

[9] For complete usage see **man 1 ipa**

```
# ipa help topics
automember          Auto Membership Rule.
automount           Automount
caacl               Manage CA ACL rules.
cert                IPA certificate operations
certprofile         Manage Certificate Profiles
config              Server configuration
delegation          Group to Group Delegation
dns                 Domain Name System (DNS)
group               Groups of users
hbac                Host-based access control commands
hbactest            Simulate use of Host-based access controls
host                Hosts/Machines
hostgroup           Groups of hosts.
idrange             ID ranges
idviews             ID Views
krbtpolicy          Kerberos ticket policy
migration           Migration to IPA
misc                Misc plug-ins
netgroup            Netgroups
otp                 One time password commands
passwd              Set a user's password
permission          Permissions
ping                Ping the remote IPA server to ensure it is running.
pkinit              Kerberos pkinit options
privilege           Privileges
pwpolicy            Password policy
radiusproxy         RADIUS Proxy Servers
realmdomains        Realm domains
role                Roles
selfservice         Self-service Permissions
selinuxusermap      SELinux User Mapping
server              IPA servers
service             Services
servicedelegation   Service Constrained Delegation
stageuser           Stageusers
sudo                commands for controlling sudo configuration
trust               Cross-realm trusts
user                Users
vault               Vaults
```

Figure 2.6: Getting list of available topics for ipa tool.

web user interface and no username and password typing is required. Figure 2.8 shows administrator being logged into the web interface.

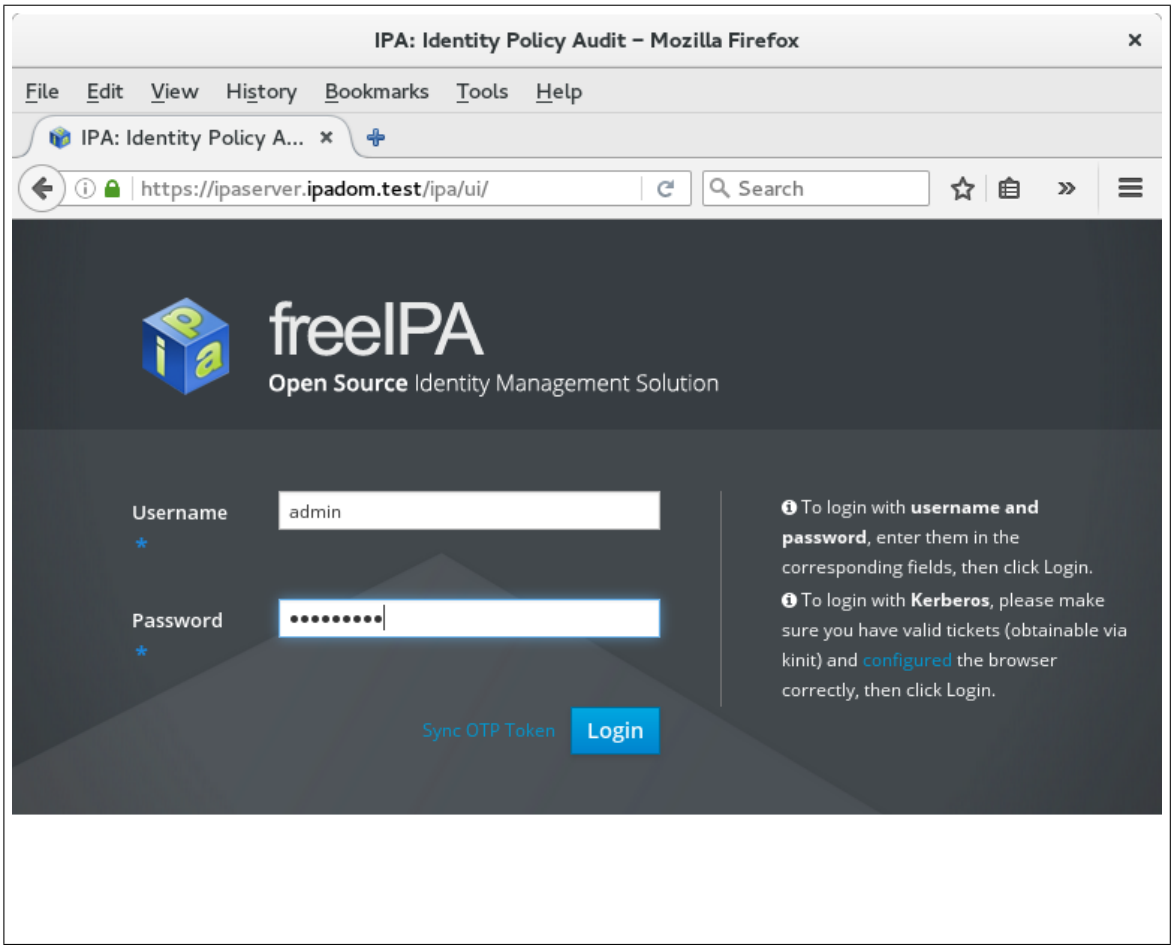


Figure 2.7: FreeIPA web UI login screen.

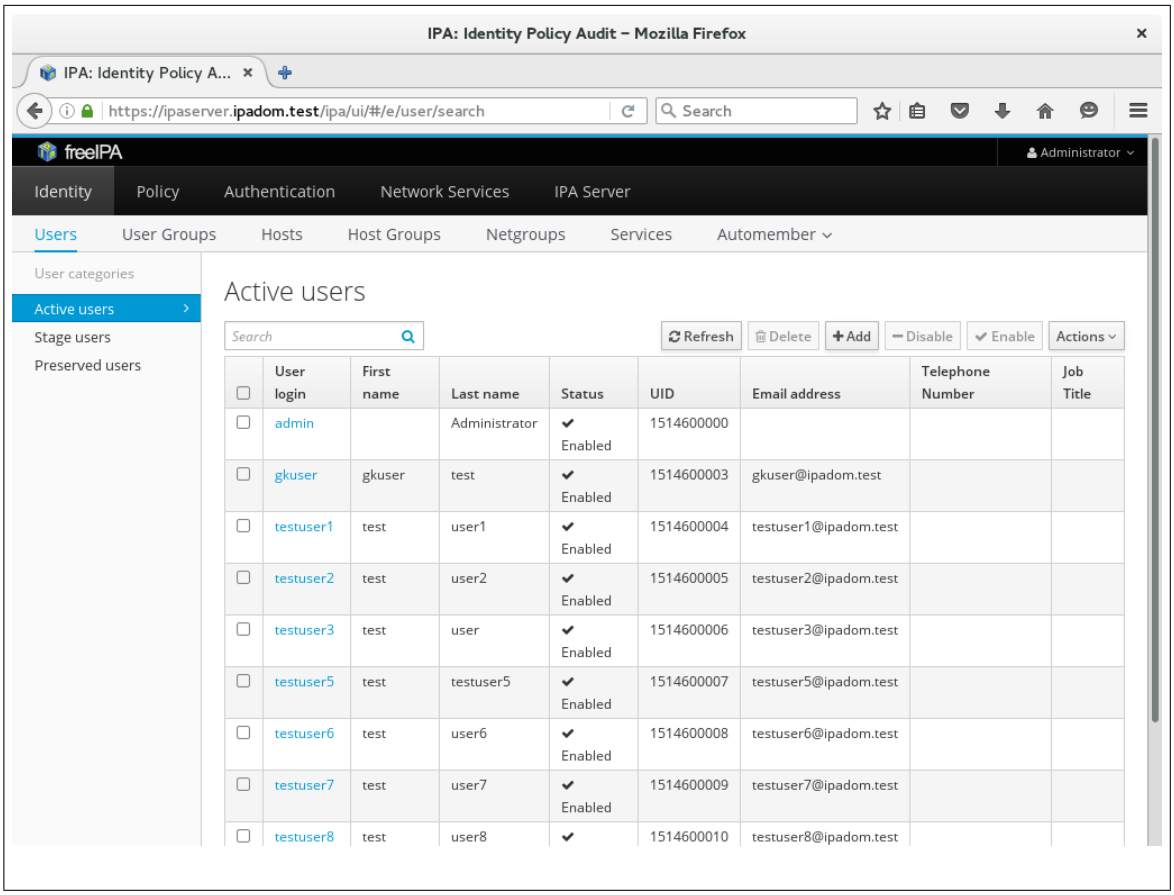


Figure 2.8: Administrator logged into FreeIPA web UI.

2.4 Enrolling a Host into FreeIPA Domain

In order to allow FreeIPA users to log into a computer other then the one where FreeIPA server is running, the computer (host) must be enrolled in FreeIPA domain and become a FreeIPA client. There are two tools commonly used for this purpose: **ipa-client-install** and **realmd**.

2.4.1 ipa-client-install Command

As the name suggest, ipa-client-install is a command that configures a host and turns it into a FreeIPA client. Similar to the ipa-server-install, it can be run in interactive mode, where the command asks a user to input all necessary options or the options can be supplied as parameters to the command. The goal is to abstract setup phase of FreeIPA client by a single command that is easy to use even for people new to FreeIPA. The command automatically detects FreeIPA server using DNS discovery and hostname of the machine that is to be enrolled in the FreeIPA domain, so it is necessary to set the hostname properly before running `ipa-client-install`. The command can automatically download FreeIPA server's CA[10] certificate and update the systemwide CA database. It configures NSS and PAM to use plugins which are able to communicate with FreeIPA through SSSD. It configures SSH to allow public key authentication for FreeIPA users (again SSSD is involved here) and modifies Kerberos configuration to join the FreeIPA Kerberos realm.

After successful run of ipa-client-install it should be possible to query the FreeIPA server for users with tools that use NSS API, for example *getent* as shown on figure 2.9 as well as use all configured SSSD services. SSSD will be discussed in more detail in chapter 3.

```
# getent passwd admin
admin:*:1686400000:1686400000:Administrator:/home/admin:/bin/bash
```

Figure 2.9: Using getent after successful client enrollment.

2.4.2 realmd Service

Program **realmd** is implemented as D-Bus service which allows callers to configure network authentication and domain membership in a standard way[18]. This makes it similar to ipa-client-install, but realmd is not limited to configure clients only as FreeIPA clients, but allows joining non-FreeIPA realms as well[11]. It internally uses ipa-client-install when joining to FreeIPA realm/domain[12]. The command line interface is provided through the **realm** tool. Figure 2.10 shows how to **discover**, **join** and then **leave** a realm using this tool.

2.5 Application Programming Interface of FreeIPA

The web interface and command line interface are not the only two ways to communicate with FreeIPA. FreeIPA allows applications to access it's features through remote Application Programming Interface (API). There are two versions of FreeIPA API. XML-RPC API,

[10] Certificate Authority
[11] For example Active Directory
[12] Realm vs domain

```
$ # Discover the realm
$ realm discover IPADOM.TEST
ipadom.test
  type: kerberos
  realm-name: IPADOM.TEST
  domain-name: ipadom.test
  configured: no
  server-software: ipa
  client-software: sssd
  required-package: freeipa-client
  required-package: oddjob
  required-package: oddjob-mkhomedir
  required-package: sssd
$ # Joining the realm will ask for admin's password
$ realm join IPADOM.TEST
Password for admin:
$ # Leaving the realm.
$ realm leave
```

Figure 2.10: Example of discovering, joining and leaving a Kerberos realm with name IPADOM.TEST with `realmd`.

which is the older version of API still used by some tools and JSON-RPC API which is newer and also provides some new functionality.

Both the `ipa` tool and the web user interface internally translate commands from user into XML-RPC or JSON-RPC request as seen on Figure 2.11. These requests are sent using HTTP protocol to one of these addresses (IPA_SERVER_HOSTNAME should be replaced with FreeIPA server hostname):
`https://IPA_SERVER_HOSTNAME/ipa/session/json`
`https://IPA_SERVER_HOSTNAME/ipa/session/xml`
for JSON-RPC and XML-RPC API requests respectively.

These requests are then processed by FreeIPA's RPC server component. After executing the method remotely, a response with results is sent back to the sender.

2.5.1 Documentation for XML ans JSON Remote Procedure Calls

Some documentation for the XML-RPC does exist, but is outdated [19]. However there is no need for new applications to use XML-RPC version of the API, because the JSON-RPC provides the same functionality and more. From now on, this document will only consider JSON-RPC API. The documentation for the JSON-RPC is currently lacking/non-existent, but there are two relatively easy ways to study the API.

The first way is to use the `ipa` tool with parameter **`-vv`** to make it more verbose. In this verbose mode it is possible to watch the request and response format with method names and arguments printed to the output [1]. The Figure 2.11 shows full JSON-RPC request taken from the output of `ipa -vv user-find admin` command. Figure 2.12 shows JSON-RPC response to this request.

Good thing to note is that the `ipa` tool command names can be mapped to the JSON-RPC method names by changing the „-“ to „_“ (dash to underscore). For example *user-find* command maps to *user_find* JSON-RPC method. This pattern works for all commands.

```
{
    "id": 0,
    "method": "user_find",
    "params": [
        [
            "admin"
        ],
        {
            "all": false,
            "no_members": false,
            "pkey_only": false,
            "raw": false,
            "version": "2.156",
            "whoami": false
        }
    ]
}
```

Figure 2.11: Example of JSON-RPC request generated by `ipa -vv user-find admin`.

Another way to study the API requires looking at the FreeIPA source code, but instead of looking at the implementation of JSON-RPC methods, a good place to start are the automated tests written for the JSON-RPC API. The tests show what results are expected when a method is called with certain parameters. This alone answers many questions a programmer might have about the API.

2.5.2 Python Bindings of FreeIPA

The *ipa* tool does not prepare the JSON/XML-RPC requests on it's own, but uses FreeIPA's Python bindings, which takes care for this task. The *ipa* tool is actually a small layer on top of Python bindings of FreeIPA. The Python bindings can be used only from FreeIPA enrolled machines. Figure 2.13 demonstrates API's used when *ipa* tool command is executed. The figure shows only JSON-RPC API as remote API, but there could be XML-RPC API used as well.

2.6 Password Vault of FreeIPA

Some programs require to work with secret data. Data that needs to be known only to certain users or programs, such as passwords, certificates, various keys etc. These sensitive data and access to them must be handled with special care so that they are not exposed to untrusted users or programs. Some programs handle storage for such data by themselves in order to have full control over them, however this can lead to errors and security vulnerabilities if the authors of such programs do not take into account all important aspects of the environment in which their programs operate. It is more robust and less error prone if the environments, such as operating systems or platforms built on top of them, already provide interfaces for secure storage. Such storage can lead to better user experience, because users do not need to type passwords over and over again. Instead the applications can store the password inside the secure storage and reuse it when needed.

```
{
    "error": null,
    "id": 0,
    "principal": "admin@IPADOM.TEST",
    "result": {
        "count": 1,
        "result": [
            {
                "dn": "uid=admin,cn=users,cn=accounts,dc=ipadom,dc=test",
                "gidnumber": [
                    "1686400000"
                ],
                "has_keytab": true,
                "has_password": true,
                "homedirectory": [
                    "/home/admin"
                ],
                "loginshell": [
                    "/bin/bash"
                ],
                "nsaccountlock": false,
                "sn": [
                    "Administrator"
                ],
                "uid": [
                    "admin"
                ],
                "uidnumber": [
                    "1686400000"
                ]
            }
        ],
        "summary": "1 user matched",
        "truncated": false
    },
    "version": "4.2.4"
}
```


Figure 2.12: JSON-RPC response to **ipa -vv user-find admin**.

This section provides information about FreeIPA's feature called **password vault** or simply **vault**, which aims to provide centralized safe storage in FreeIPA domain with focus on how this feature is exposed to users or applications via it's API. This feature was first introduced in FreeIPA version 4.2.

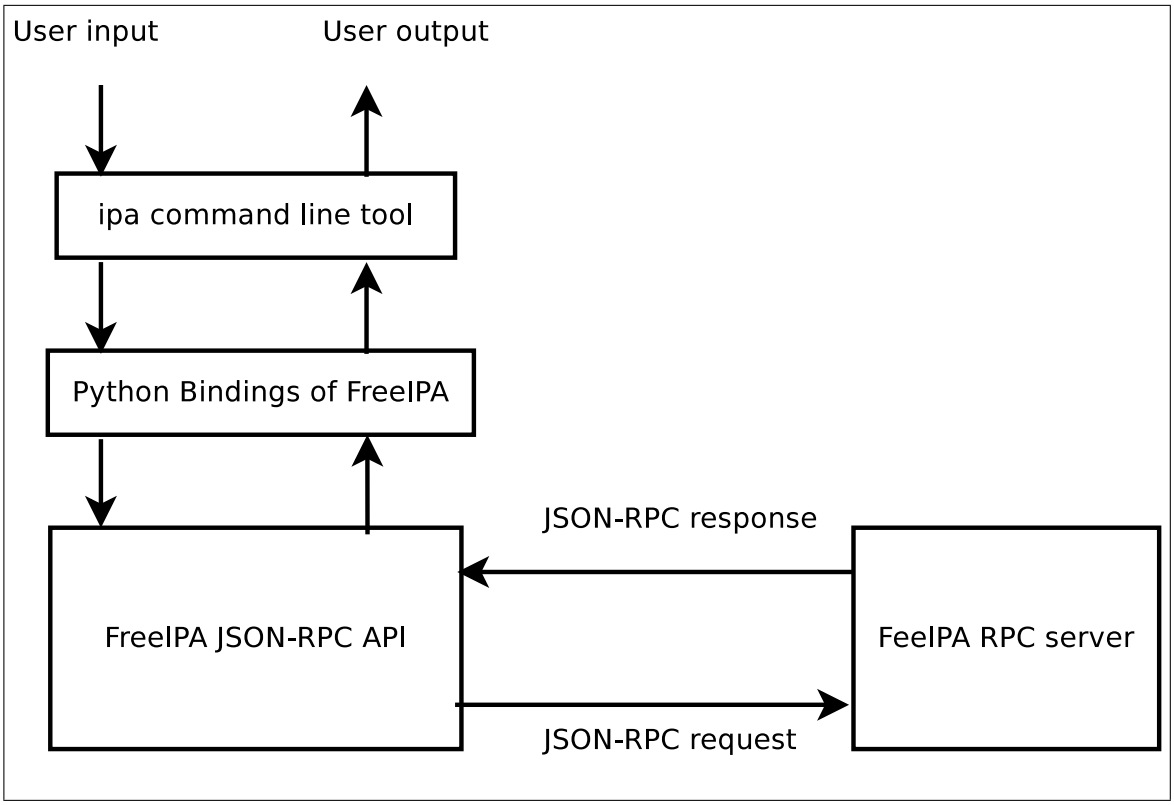


Figure 2.13: API's in use when `ipa` tool is invoked.

2.6.1 Overview of Password Vault

Password vault is a feature that allows FreeIPA users to keep secret data inside **vaults**, that are stored centrally in LDAP backend using FreeIPA's component **KRA**[13]. Vault can be owned by user, service or can be shared among users. Single user or service can own multiple vaults. Secret stored inside a vault can be any binary data[21]. This way, users and services do not keep their sensitive data spread across multiple hosts in FreeIPA domain, but use one central storage and access it from any enrolled host.

The KRA component is optional and must be installed additionally. After installing FreeIPA server using *ipa-server-install*, it is possible to add KRA component with **ipa-kra-install** [21].

Design document and features of released and planed versions for Password Vault are available on FreeIPA's wiki page [20].

2.6.2 Storing and Retrieving Secrets with KRA

The store operation (illustrated on 2.14) is initiated by client. First, the client generates a random **session key** and uses it to encrypt the secret. The client then encrypts both the session key and the secret by using **KRA's transport public key** and sends them to KRA. When KRA receives these encrypted data it uses it's **private transport key** to decrypt the session key and finally uses the session key to decrypt the data and stores it to the LDAP backend where it is encrypted with **storage key**[21].

[13]Key Recovery Agent

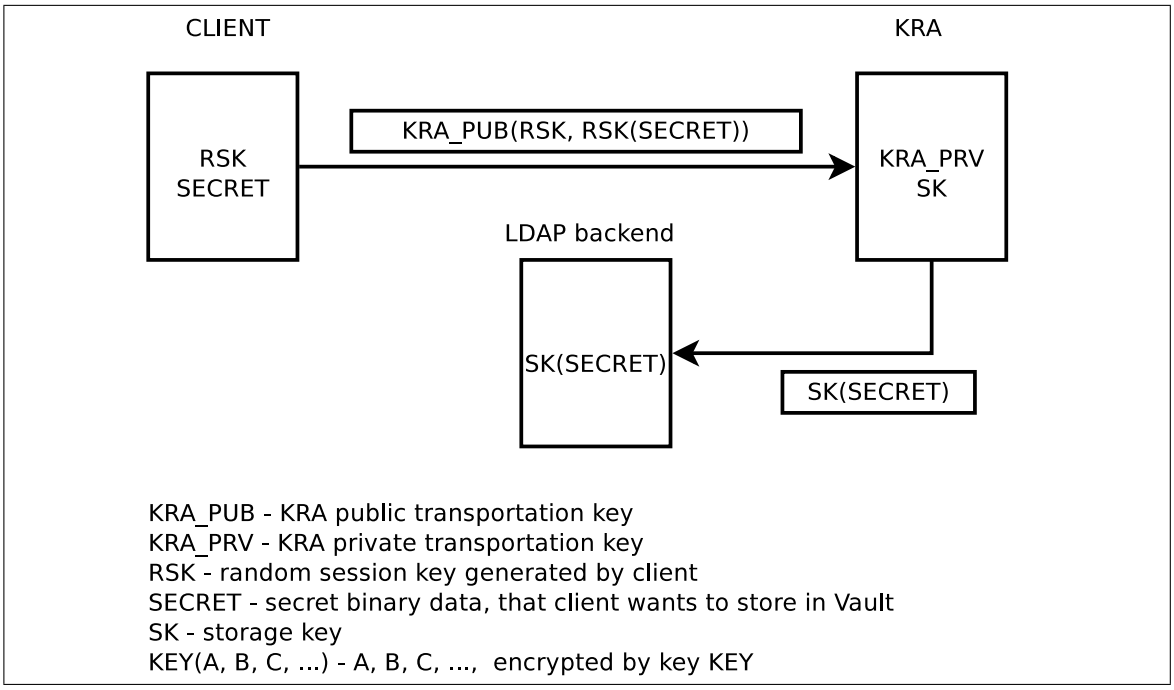


Figure 2.14: Client stores data in KRA.

When a client[14] wants to get the secret back, it again generates a random **session key**, encrypts it with the **KRA's transport public key** and sends it to KRA. KRA decrypts the session key with it's **private transport key**. KRA retrieves the secret from LDAP backend and decrypts it using **storage key**. Finally it encrypts the secret with the session key and sends it to client [21].

2.6.3 Types of Vaults

There are three different types of vaults: standard, symmetric and asymmetric.

The **standard** vault only depends on the transportation mechanism described in 2.6.2. No additional password is needed and any authorized vault owner can read or write secrets in the vault [21].

In the case of **symmetric** vault the secret is additionally encrypted with a shared symmetric key. Users have to encrypt the secrets before sending them to the vault and decrypt them after they receive them from the vault [21].

Assymetric vault protects secrets with additional asymmetric key pair. Klients use the public key when they store secrets and private keys when they retrieve them. The public key is available to all vault users, but the private key only to the owners [21].

Vaults are stored internally in so called containers. There are currently three different containers available for three different ownership categories: private, service and shared vaults. Private vaults are owned by users, service vaults by services and shared vaults by admin, who can make them available to other users.

[14] Kerberos authenticated FreeIPA user.

2.6.4 User Interface for Manipulating Vaults

To manage vaults a command line interface integrated with the *ipa* tool is provided. Examples can be found in the wiki [20] or in the vault topic help, that can be browsed by **ipa vault help**. The ipa tool's help will also list all available commands for the topic, which can be shown on figure 2.15.

```
Topic commands:
  vault-add                     Create a new vault.
  vault-add-member              Add members to a vault.
  vault-add-owner               Add owners to a vault.
  vault-archive                 Archive data into a vault.
  vault-del                     Delete a vault.
  vault-find                    Search for vaults.
  vault-mod                     Modify a vault.
  vault-remove-member           Remove members from a vault.
  vault-remove-owner            Remove owners from a vault.
  vault-retrieve                Retrieve a data from a vault.
  vault-show                    Display information about a vault.
  vaultconfig-show              Show vault configuration.
  vaultcontainer-add-owner      Add owners to a vault container.
  vaultcontainer-del            Delete a vault container.
  vaultcontainer-remove-owner   Remove owners from a vault container.
  vaultcontainer-show           Display information about a vault container.
```

Figure 2.15: List of vault topic commands retrieved by **ipa vault help**.

```
# ipa vault-add vault1 --type=standard
--------------------
Added vault "vault1"
--------------------
  Vault name: vault1
  Type: standard
  Owner users: admin
  Vault user: admin
# ipa vault-archive vault1 --data=c2VjcmV0MTIz
---------------------------------
Archived data into vault "vault1"
---------------------------------
# ipa vault-retrieve vault1
-----------------------------------
Retrieved data from vault "vault1"
-----------------------------------
  Data: c2VjcmV0MTIz
```

Figure 2.16: Create vault, archive and retrieve data.

The figure 2.16 shows how new private vault is added and how secret is stored and retrieved from it. The secret can be passed through command line arguments directly or as

a file. When passing the secret through command line directly as shown on figure 2.16, the secret must be encoded with base 64. The retrieved secret is also base 64 encoded. When using files, the base 64 encoding is not needed on input, but internally it will still be base 64 encoded when passed to the RPC server through JSON-RPC request.

Chapter 3

System Security Services Daemon

This chapter provides information about System Security Services Daemon (SSSD) and what role it plays in relationship with FreeIPA. Internal architecture and some features of SSSD will also be discussed in this chapter.

3.1 Overview of SSSD

SSSD is a daemon that provides data about identities managed by remote resource such as FreeIPA or LDAP. It provides NSS (identity) and PAM (authentication) modules to client applications[1]. There is also a D-Bus interface being developed that can provide further information to applications. SSSD stores data into local filesystem cache and reuses it to improve response times. This local cache can also be used in case when remote resource is not available, for example users will be able to log into hosts while being offline. SSSD started as a sister project of FreeIPA and development of both is closely related. SSSD is capable to act as a client to plain LDAP server as well as Active Directory (directory service for Microsoft Windows Server) and also manage local users created directly by SSSD in so called LOCAL domain. This chapter focuses on SSSD as FreeIPA's client side component.

3.2 About NSS and PAM

This section will briefly introduce NSS[2] and PAM[3] technologies.

3.2.1 Name Service Switch

Information about users in UNIX-like systems is traditionally kept in */etc/passwd* file and information about groups in */etc/groups* file. Each line in */etc/passwd* file represents a different user, for example the line with user root looks like this:

```
root:x:0:0:root:/root:/bin/bash
```

The line tells us the users name, UID and GID numbers, GECOS field, which is a field that can contain any additional information about the user or can be left blank, then the users home directory and finally users login shell. Similar, the */etc/group* file contains lines

[1] In relationship with applications on the host, SSSD is in the role of a server
[2] Name Service Switch
[3] Pluggable Authentication Module

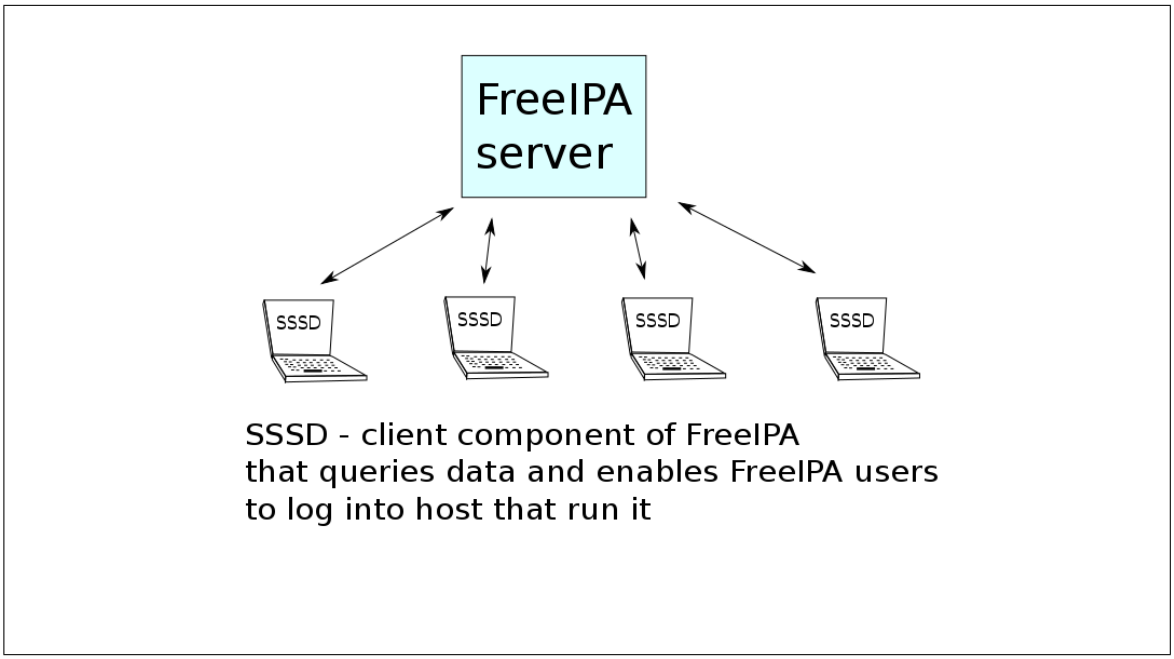


Figure 3.1: FreeIPA server and SSSD clients.

where each line provides information about one group. For example the group wheel can look like this in:

```
wheel:x:10:user
```

The line shows us the group's name, GID and users, that are members of this group. More information about these files can be obtained from manual pages.

These files are readable by anyone on the host, so if an application needs to access information about users or groups, it can parse these files and get that information. However this is not a correct solution for modern systems, because these files are not the only providers of these information and there can be other sources of group and user information, for example FreeIPA stores this information in it's LDAP component and you will not find FreeIPA users or groups in */etc/passwd* or */etc/groups*.

One of the solution's for this problem is the NSS, Name Service Switch, which is mechanism implemented in the GNU C library inspired by method used in the C library for Solaris 2 from Sun Microsystems, where also the name NSS comes from[2].

The main idea is to provide unified API to access databases with information about groups, users and some other type of databases (such as netgroups, mail aliases etc.) and allow to write plugins for NSS, where each plugin needs to implement the NSS API and based on configuration in */etc/nssswitch.conf*, the correct plugin is used to obtain required information. The configuration file also specifies the order in which these plugins are tried until the requested data is found. In case of SSSD, the plugins name is „sss“. Figure 3.2 shows content of /etc/nssswitch.conf on FreeIPA enrolled client that uses SSSD. The „files“ plugin is plugin to access */etc/passwd* and */etc/groups* and it has higher priority than the „sss“ plugin, which means it is searched first. We can see that SSSD provides passwd, shadow, group, services, netgroup, automount and sudoers databases. Which plugin is used is completely transparent for the applications that use NSS API.

```
passwd:     files sss
shadow:     files sss
group:      files sss
hosts:      files mdns4_minimal [NOTFOUND=return] dns myhostname
bootparams: nisplus [NOTFOUND=return] files
ethers:     files
netmasks:   files
networks:   files
protocols:  files
rpc:        files
services:   files sss
netgroup:   files sss
publickey:  nisplus
automount:  files sss
aliases:    files nisplus
sudoers: files sss
```

Figure 3.2: Content of /etc/nssswitch.conf on FreeIPA enrolled client.

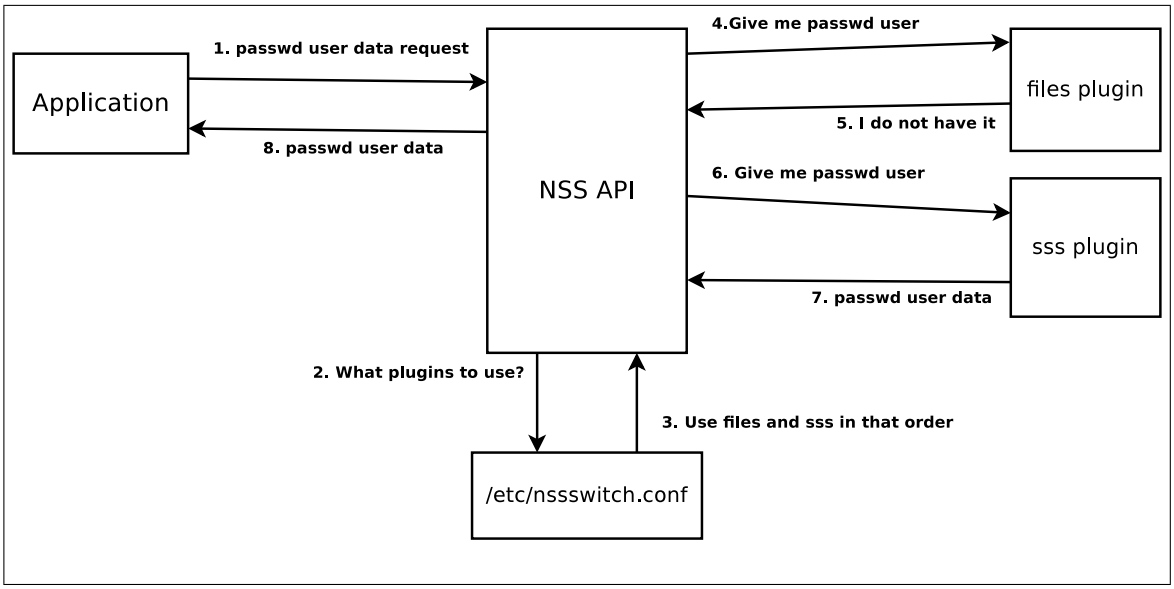


Figure 3.3: Application asks for passwd user through NSS.

3.2.2 Pluggable Authentication Module

While NSS library abstract from applications how data from some specific types of databases are retrieved, PAM library does similar job in a different area. It abstracts authentication management, session management, account management and password management[3]. There are several implementations of PAM libraries. The one used on systems with Linux kernel is called Linux-PAM, but all PAM library implementations share the same concepts.

System administrator can specify preferred mechanism for authentication, session, account and password management in any PAM aware application by specifying a PAM modules stack in the appropriate PAM configuration file. There can be several PAM configuration files for different PAM aware applications usually located in the */etc/pam.d/*

directory[4]. For example if user is to be authenticated in a PAM aware application, the actual authentication takes place in dynamically loaded modules that are specified in the PAM stack for the authentication. In the PAM stack, it is possible to specify which modules are required to succeed in order for the authentication to be successful, or if it is sufficient to succeed only in some. Similar to the NSS, the modules are loaded in the order specified in PAM configuration, however the authentication may not end with the first successful module.

SSSD's PAM module needs to be specified in the PAM stack for all PAM aware application in order to allow authentication of SSSD users. SSSD's PAM module can also be used for password, session and account management PAM stacks. Figure 3.4 shows part of PAM configuration file *system-auth* on FreeIPA enrolled client. This part configuration snippet shows how *pam_sss* module was included in the authentication (auth) stack. In fact it is included in all PAM stacks, but they are not shown on the Figure **??**.

```
auth        required      pam_env.so
auth        sufficient    pam_fprintd.so
auth        [default=1 success=ok] pam_localuser.so
auth        [success=done ignore=ignore default=die] pam_unix.so nullok try_
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        sufficient    pam_sss.so forward_pass
auth        required      pam_deny.so
```

Figure 3.4: SSSD's PAM module pam_sss in authentication PAM stack.

3.3 SSSD Architecture

SSSD deamon consists of several processes that each play a different role in the system.

The first process that is started is called **monitor** (not shown in the figure). This process starts other processes and periodically checks if they are alive by sending them small messages (pings) via socket file and waits for the answer. If the answer does not come in a certain amount of time, monitor assumes there is some problem with the process and terminates it as shown on Figure 3.5. Then the process is started again by the monitor. There are some other tasks that monitor is responsible for, but most of the time, it is relatively passive process.

Depending on how SSSD is configured monitor starts one or multiple **backend** processes, one for each configured domain to which SSSD connects. This indicates that SSSD can connect to multiple domains and query data from multiple remote resources, but this feature is mostly used only to cover some corner cases, like when the remote servers can not form a trust relationship and most of the installation have SSSD connected to only one domain. At least one domain must be configured otherwise SSSD refuses to start. In the case on the Figure 3.6 there is one backend process configured to join remote FreeIPA server with IPA plugin.

Another processes that are started by monitor are so called **responders**. Responders are processes implementing certain services that SSSD provides. The reason why they are called responders is that they talk directly to applications that use their interface (they respond to requests). Two responders are configured in the example on Figure 3.6. The

[4]The location of this directory can differ on various operating systems.

PAM responder and the **NSS responder**. We will talk more about these responders later in this chapter.

3.3.1 Example of SSSD Operation

A more detailed (but still simplified) look at the SSSD architecture will be provided by a following example operation in which application using NSS API asks for information about FreeIPA user. This could be for example the case of entering *getent passwd admin* in the console on FreeIPA enrolled client. For this to work the *passwd* line in /etc/nsswitch.conf needs to contain *sss* after *files*, but this configuration change is done automatically by ipa-client-install script, so if the client was enrolled in a standard way, it should just work.

First the client NSS application loads the **sss** variant of NSS module. Some results of NSS requests are cached in so-called **memcache**[5]. If this is that type of request, the memcache is consulted first (see step 0 on Figure 3.6). If requested entry is in the memcache and is valid (not expired) then NSS responder is not contacted at all and the client application reads the data directly from memcache. This significantly improves SSSD's performance. Memcacahe is specific to NSS only (PAM or other services do not have it).

If the memcache did not provide the requested entry, then the client application contacts SSSD's **NSS responder** using a private binary protocol (step 1 on Figure 3.6). NSS responder consults the SSSD's local databese (step 2 on Figure 3.6. If the requested data is available it is send back to the client application. If not, NSS responder creates and sends a request to the **backend** (step 3 in 3.6) and waits for backend's reply. Backend can be configured to use different plugins to contact the remote resource. The example on on Figure 3.6 shows the IPA plugin being used to contact remote FreeIPA's LDAP component and query it for data (step 4 Figure 3.6). Then the **backend** process updates the local database with new data (step 5 on Figure 3.6). When local database is updated, the backend process notifies NSS responder about the result (step 6 in 3.6). NSS responder takes the new data from local database (step 7 in 3.6) and finally sends them to the client application (step 8 in 3.6).

[5]Memcache is also known as memory cache, mmap cache, fastcache or fast in-memory cache.

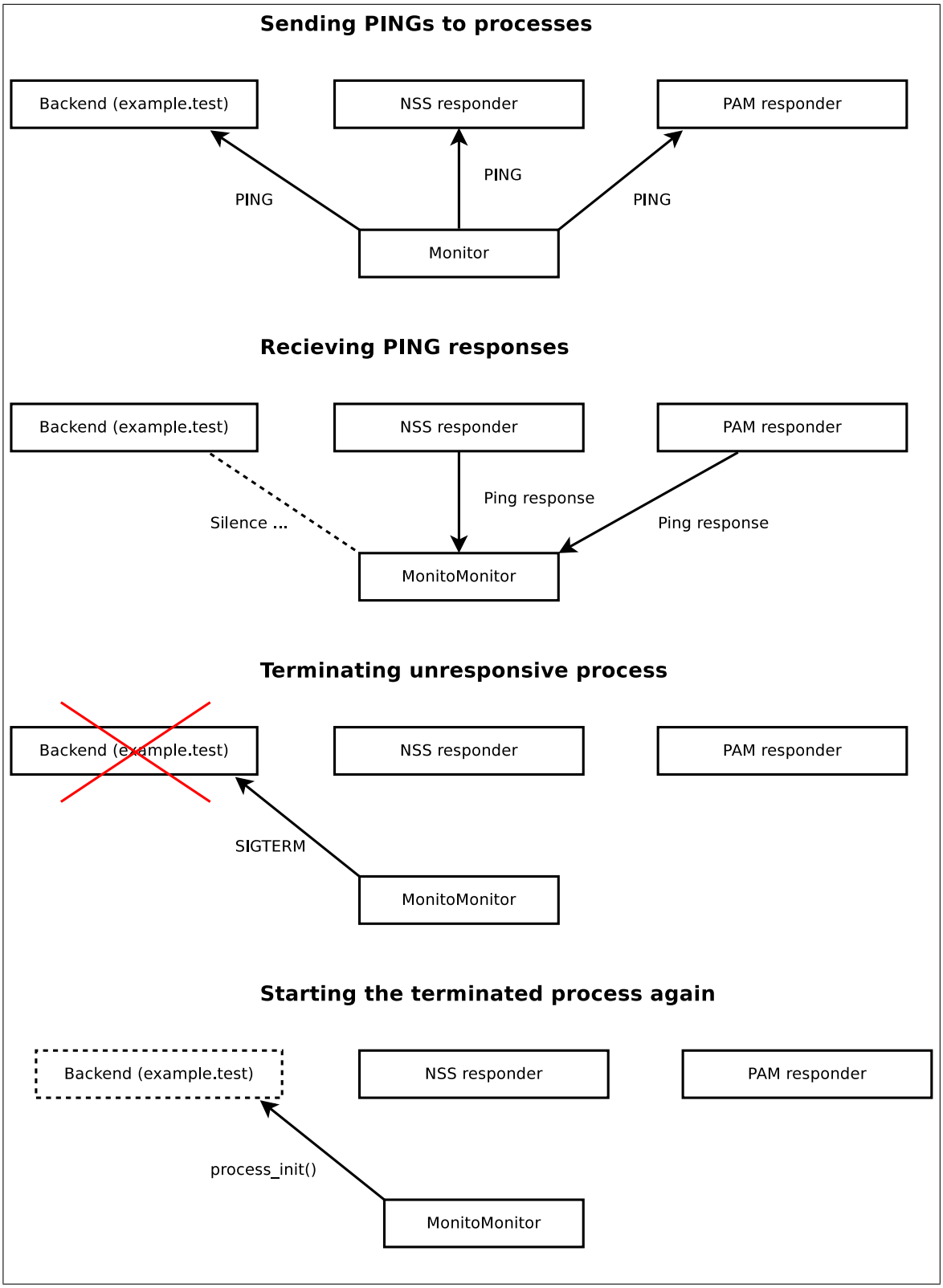


Figure 3.5: Monitor process controlling other SSSD processes and terminating unresponsive process.

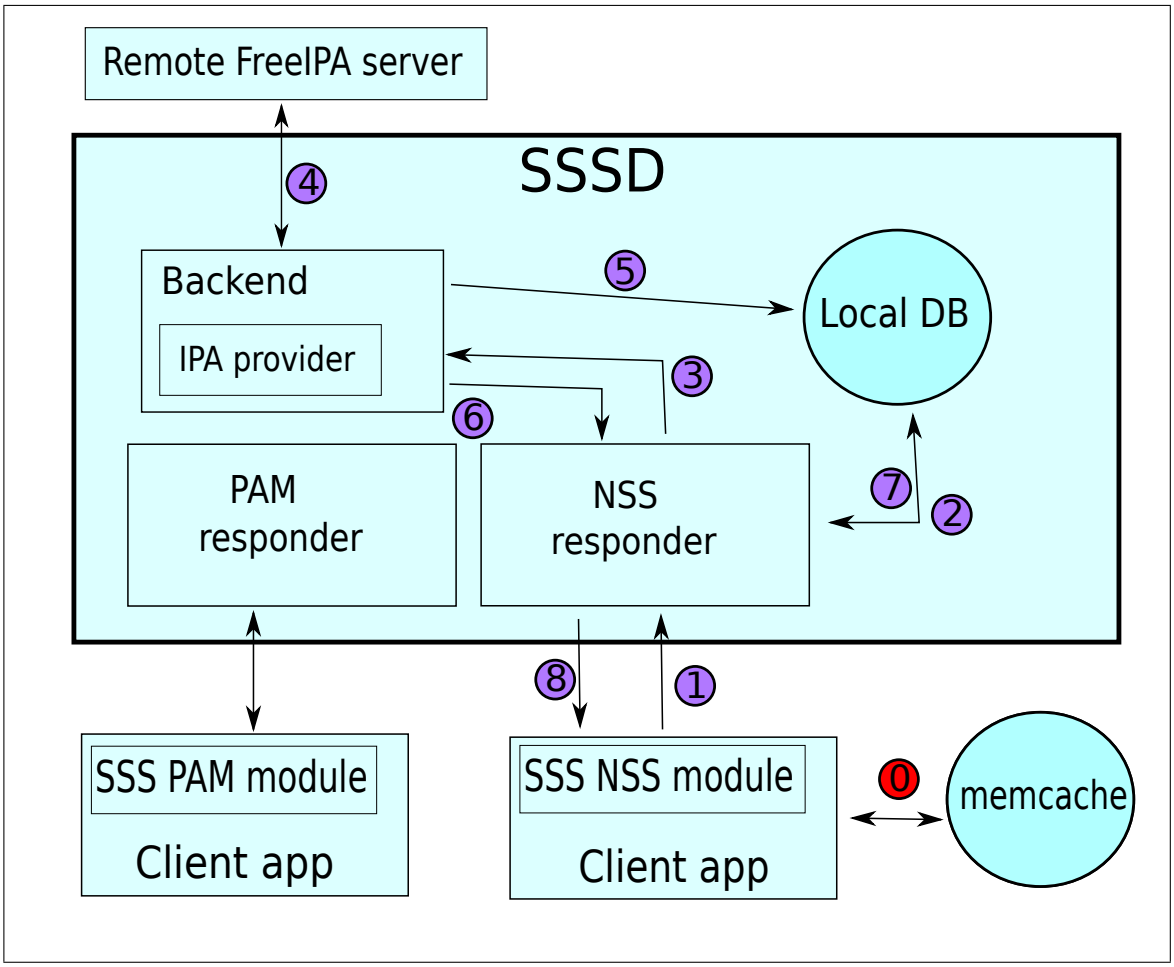


Figure 3.6: Simplified SSSD architecture.

Chapter 4

GNOME Keyring

This chapter provides information about GNOME Keyring project and it's architecture in Section 4.1. Section 4.2 gives overview of Secret Service API, which is API provided by GNOME Keyring.

4.1 Overview of GNOME Keyring

GNOME project is led by non-profit GNOME Foundation[12]. GNOME developers are involved in development of many different free software pieces that together aim to provide a complete software solution[13].

GNOME Keyring is one of GNOME's projects. It is a collection of components that together provide storage for secrets like passwords, keys and certificates. It offers easy to use API for application developers to access it's features.

GNOME Keyring uses PKCS#11, a Public Key Cryptography Standard which is often used to manage secure storage in smart cards [10]. PKCS#11 is an open standard. More information about it can be found on OASIS[1] PKCS#11 website [17].

GNOME Keyring stores secrets in *keyrings* which are encrypted with master password and stored on disk. User can own several *keyrings* for different purposes. The default *keyring* is called *login* and the password for it is usually the same as user's login password. There is also one special *keyring* that is never stored on disk. This *keyring* is called *session* and information in it is lost when user logs out and his session ends [14]. However this terminology is being replaced by the Secret Service API terminology. More about it later in this chapter.

4.1.1 Architecture of GNOME Keyring

GNOME Keyring consists of multiple components. GNOME Keyring's wiki page [9] provides basic information about these components and used architecture, that can be seen on Figure 4.1.

The figure shows that GNOME Keyring provides PAM module. The PAM module has name ***pam_gnome_keyring.so*** and provides functionality for three PAM categories: authentication management module, session management module and password management module[11].

[1] OASIS - Organization for the Advancement of Structured Information Standards.[16]

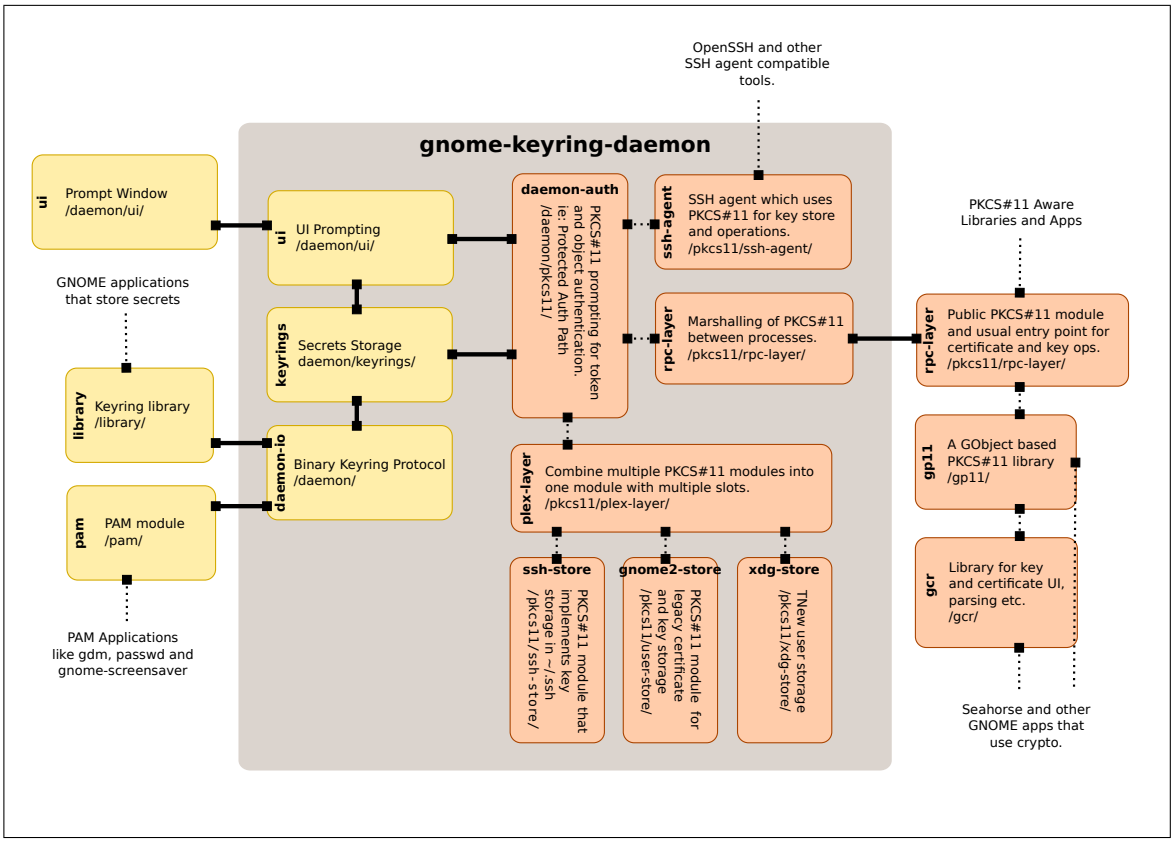


Figure 4.1: GNOME Keyring components [9].

The role of **authentication module** is to store the password from previous module in the PAM stack for later use. If option *auto_start* was passed to the module, it will automatically start GNOME Keyring deamon and unlock the *login keyring* with the retrieved password [11].

The **session management** module unlocks the *login keyring* if the deamon is running and if the authentication module successfully retrieved password. This module also provides functions to start and terminate sessions. When terminating a session it will also terminate the deamon if the deamon was started in either session management module or authentication module. The option *auto_start* can be passed to this module as well. The option has the same meaning and it must be passed to one of these two modules otherwise the deamon will not start. [11]

Password management module allows to change the password for *login keyring*. It uses password passed by previous module in the PAM stack [11].

Figure 4.2 shows example of PAM configuration for login into *GDM*[2]. As you can see the password management module is not used in this example. This configuration will start the daemon in the session management module. After user logs in using *GDM*, the *login keyring* will be unlocked and ready to use until user's session ends.

There are some differences between the architecture in figure 4.1 and the current actual architecture of GNOME Keyring. The module *keyrings* shown on the figure was used to manage the keyrings. It was connected to the *daemon-io* that communicated with the

[2]The GNOME Display Manager (GDM) is a program that manages graphical display servers and handles graphical user logins [8]

auth	required	pam_unix.so
auth	**optional**	**pam_gnome_keyring.so**
account	include	login
session	include	login
session	**optional**	**pam_gnome_keyring.so auto_start**
password	include	login

Figure 4.2: Example of /etc/pam.d/gdm. Source https://wiki.gnome.org/Projects/GnomeKeyring/Pam/Manual

library[3] used by applications via private protocol. This is no longer the case. The *keyrings* module was replaced with a new DBus module and the communication with applications or libraries no longer happens through a private protocol, but through a well defined DBus API called Secret Service API. More about this API is written in the next section. The original *library* can still be used. It is called *libgnome-keyring*, but it is being replaced by new library called *libsecret* which was designed for the Secret Service API from the very beginning which makes the *libgnome-keyring* deprecated. Both of these libraries currently use the Secret Service API internally.

With the switch to Secret Service API a new PKCS#11 storage has been added. It is called *secret-store*. This also not shown on the figure 4.1, but it would be at the same level as the *ssh-store gnome2-store* and *xdg-store*.

Figure 4.1 also shows that GNOME Keyring provides interface dor PKCS#11 aware applications. The components for this are seen on the right side of the figure.

4.2 Secret Service API

GNOME Keyring and KDE Wallet[4] both used in the past their own private APIs to store secrets. Secret Service API is result effort taken by GNOME and KDE developers to create a new unified API for managing secrets and use it in both environments[4]. This helps to increase portability of applications between GNOME, KDE and other environments that may provide DBus service with this API. The API allows applications to create **collections** that can hold **items** in which secrets can be stored. This API defines three new DBus interfaces:

- org.freedesktop.Secret.Service,
- org.freedesktop.Secret.Collection, and
- org.freedesktop.Secret.Item.

The **org.freedesktop.Secret.Service** is interface for the actual service. It provides methods for opening session for applications, creating new collection where items can be stored, locking and unlocking items and collections, searching for items in any collection, getting the secrets from any collection and setting aliases for collections [5].

The **org.freedesktop.Secret.Collection** is interface of a collection created by the *org.freedesktop.Secret.Service.CreateCollection* method call. It allows creating new items inside collection and searching for these items [5].

[3] Library written in C
[4] Safe storage in KDE environment

The **org.freedesktop.Secret.Item** is interface of an item created by *org.freedesktop.Secret.Collection.CreateItem* method call. It allows store the actual secret in the item and then read it [5].

Chapter 5

Integrating GNOME Keyring with FreeIPA

This chapter will provide in Section 5.1 information about benefits of integrating GNOME Keyring and FreeIPA. In Section 5.2, different high level designs of possible integration methods will be presented. One of these designs was chosen for implementation of a prototype solution.

5.1 Motivation for Integration FreeIPA and GNOME Keyring

GNOME Keyring and FreeIPA's password vault are both implementations of safe storage for secrets. GNOME Keyring provides local storage for the host, while password vault provides centralized storage for all FreeIPA clients. Some users may wish to store their secrets from GNOME Keyring (or Secret Service API) aware applications centrally on FreeIPA server. This would make their GNOME Keyring secrets available to all hosts in the FreeIPA domain. If GNOME Keyring was integrated into FreeIPA's password vault, the users could benefit from centrally stored and managed secrets, while application developers could keep using GNOME Keyring/Secret Service API and do not introduce additional changes into their programs. In this chapter we will consider the integration of the part of GNOME Keyring that is responsible for dealing with request made to the Secret Service API. This covers applications that use either *libgnome-keyring* or *libsecret* libraries, or talk directly to the Secret Service API.

5.2 Possible Ways to Integrate GNOME Keyring with FreeIPA Vault

There are several ways the integration could be solved. Depending on where the communication with FreeIPA takes place, it is possible to split them into three main categories:

- solutions where GNOME Keyring daemon directly communicates with FreeIPA,
- solutions with new SSSD service that is exposed for GNOME Keyring daemon and handles the communication with FreeIPA, and
- solutions that bypass GNOME Keyring daemon and communicate with FreeIPA from client libraries.

5.2.1 Talking to FreeIPA Directly from GNOME Keyring

One of the possible ways to integrate GNOME Keyring with FreeIPA is to use FreeIPA's JSON-RPC API for password vault management directly from GNOME Keyring daemon. There are several places in GNOME Keyring daemon where the communication could be initiated from. Each would require implementation of new module that would abstract the communication with FreeIPA. Details about this module will be provided in chapter 5.4.

One possible way is to implement a new PKCS#11 storage module. The communication would take place from this module and the GNOME Keyring daemon would have to switch between this module or the *secret-store* module based on configuration. Figure 5.1 demonstrates this approach.

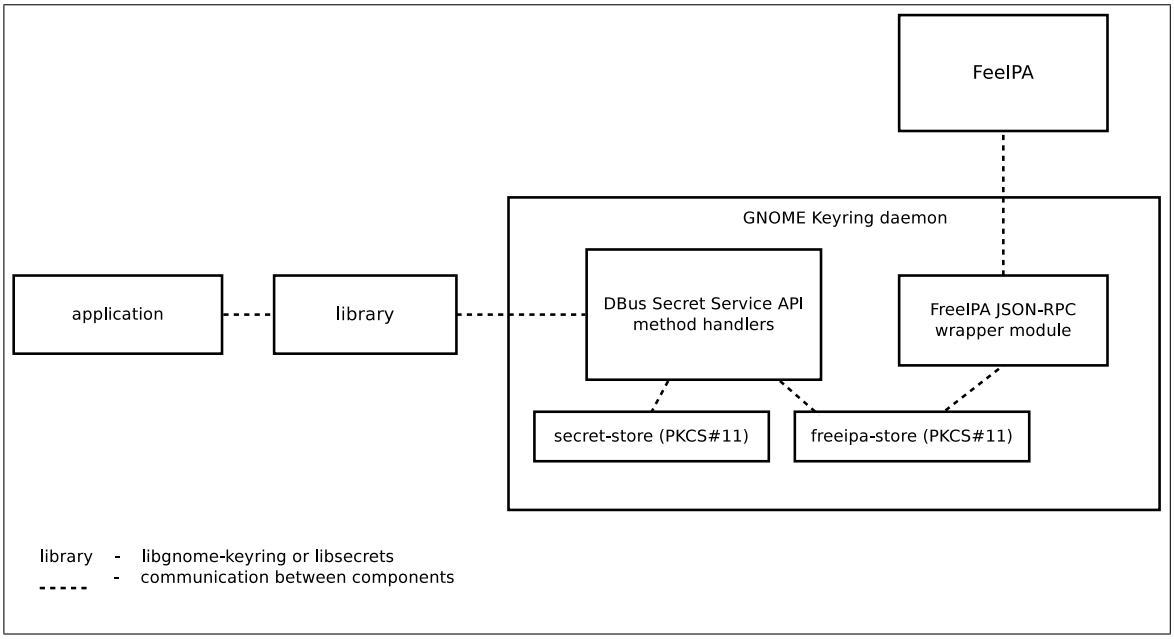


Figure 5.1: Added PKCS#11 module that uses the new FreeIPA JSON-RPC wrapper module

Alternatively the existing PKCS#11 *secret-store* module could be modified to conditionally route the secrets to FreeIPA or to use local storage (or both). This is demonstrated on Figure 5.2.

Another approach would be to operate on a higher level in the daemon and make the DBus module that implements the Secret Service API method handlers conditionally choose between using a newly implemented module to route the secrets to FreeIPA's vault, in which case the secret would not be stored in any of the PKCS#11 modules, or use the PKCS#11 storage as usual. This would require modifications in the method handlers. Figure 5.3 demonstrates this approach.

The main role of SSSD in all the above concepts would be to propagate FreeIPA server hostname to the GNOME Keyring daemon. This could be done in several ways and is discussed more in Section 5.4.

Advantages All the above approaches share the same advantages and disadvantages. The initial implementation is relatively simple in comparison to implementing new SSSD responder approach which allows to create a prototype sooner and experiment with the

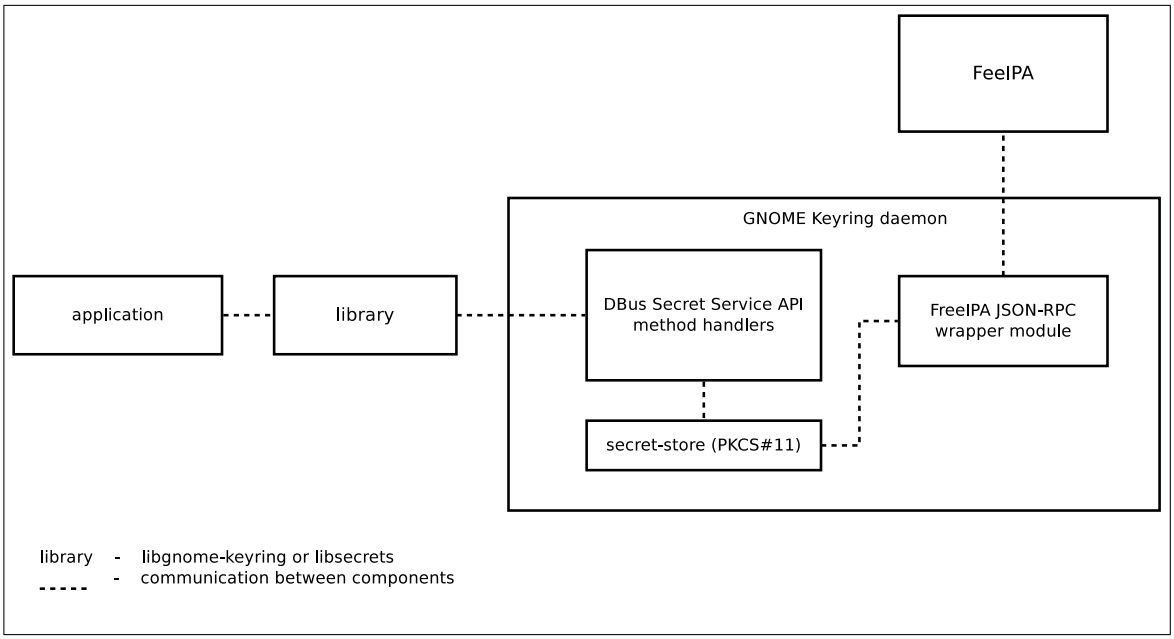


Figure 5.2: Module secret-store uses the new FreeIPA JSON-RPC wrapper module

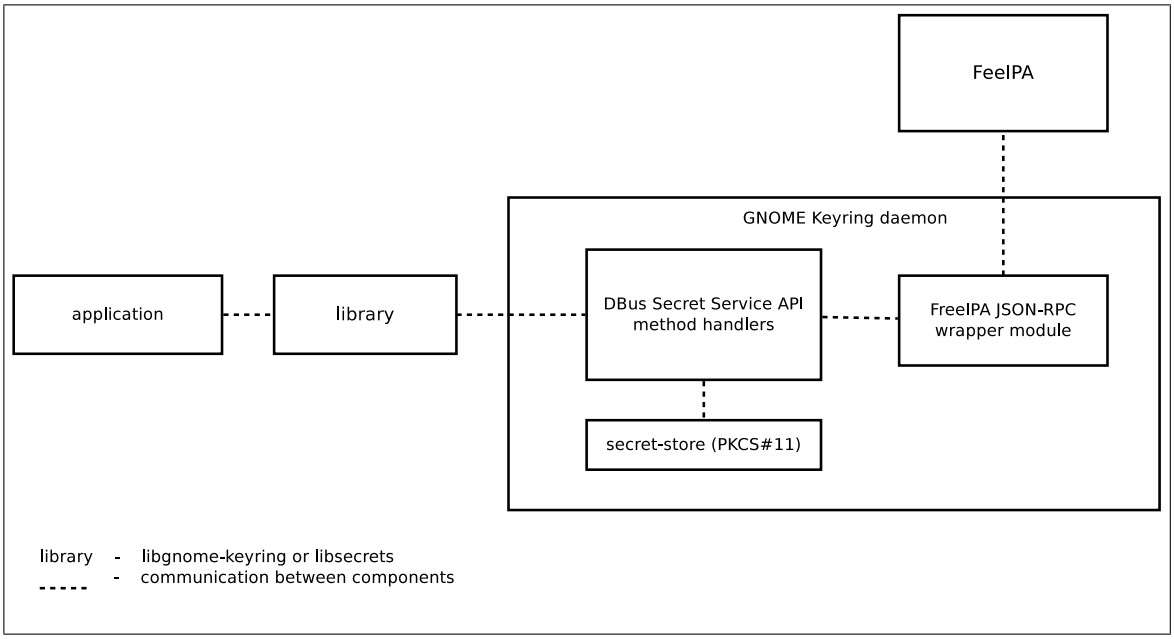


Figure 5.3: DBus module initiates communication through the new FreeIPA JSON-RPC wrapper module

integration. This is also why this type of approach was chosen for the implementation of the prototype that is described later in this chapter.

Disadvantages The main disadvantage is that it looses the benefit of being part of SSSD which would allow for easier access for FreeIPA information and provide infrastructure to deal with the remote resource. This disadvantage would grow together with the number of features implemented.

5.2.2 Adding New SSSD Service

Another way to approach the integration would be to implement new responder service in SSSD. This service would provide library to handle communication between the responder and it's clients. The client would be GNOME Keyring daemon. The question, „Where to initiate the communication with the service responder?“ is similar to the question „Where to initiate communication with the FreeIPA?“, in the above approaches. The communication with FreeIPA would be than completely handled by this new responder.

Advantages SSSD as a client side component of FreeIPA is the natural place for this kind of tasks. SSSD has all the necessary information about FreeIPA to start the communication without the need to expose additional information to third party programs such as GNOME Keyring daemon. SSSD could also use it's local storage to cache the secrets for offline use. This approach is demonstrated in Figure 5.4.

disadvantages The main disadvantage is the actual need to implement the responder which is currently not a trivial task and would require significantly more time than any of the already mentioned approaches.

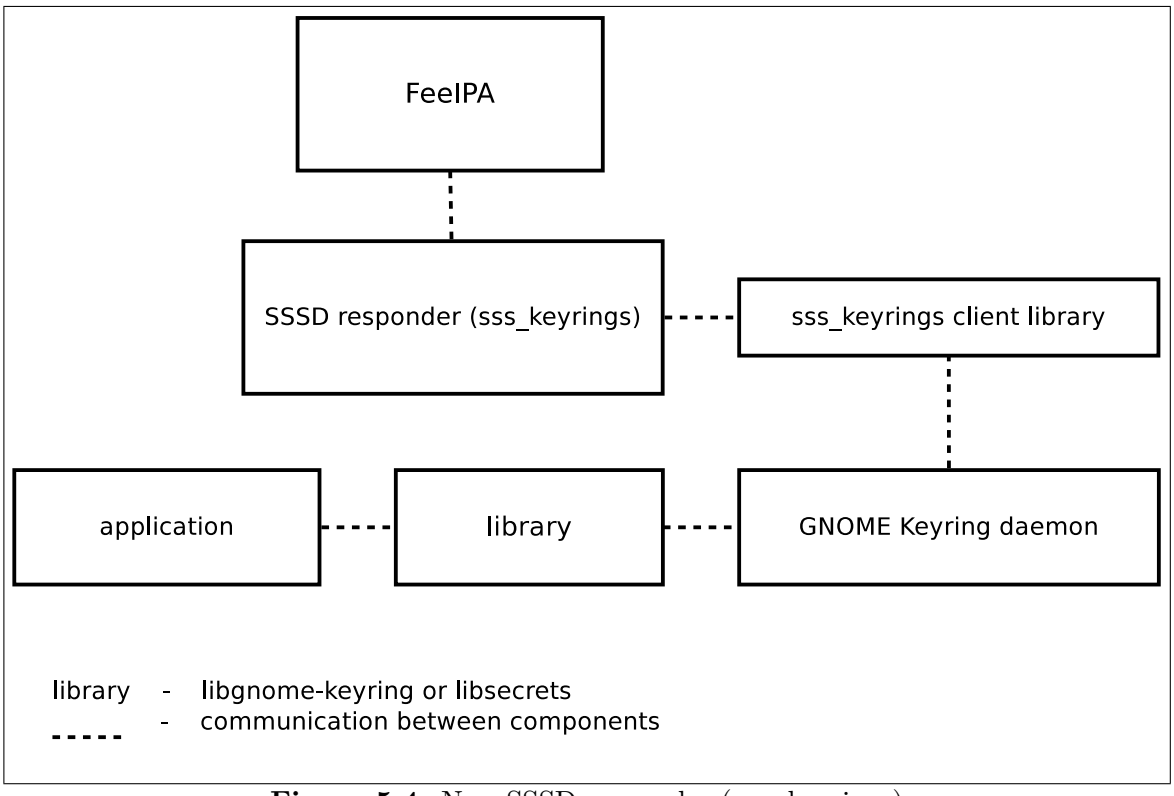


Figure 5.4: New SSSD responder (sss_keyrings)

5.2.3 Avoiding GNOME Keyring Daemon

There is one more interesting way to implement the integration without touching the GNOME Keyring daemon at all and it is actually very simple one. The communication could take place from the *libgnome-keyring* or *libsecrets* libraries directly. This approach is demonstrated in Figure 5.5.

Advantages This would be the easiest way to do the initial version with minimal set of features.

Disadvantages Applications that directly use the Secret Service API would not be covered by this approach. In the end it would basically require implementing parts of GNOME Keyring daemon inside the libraries.

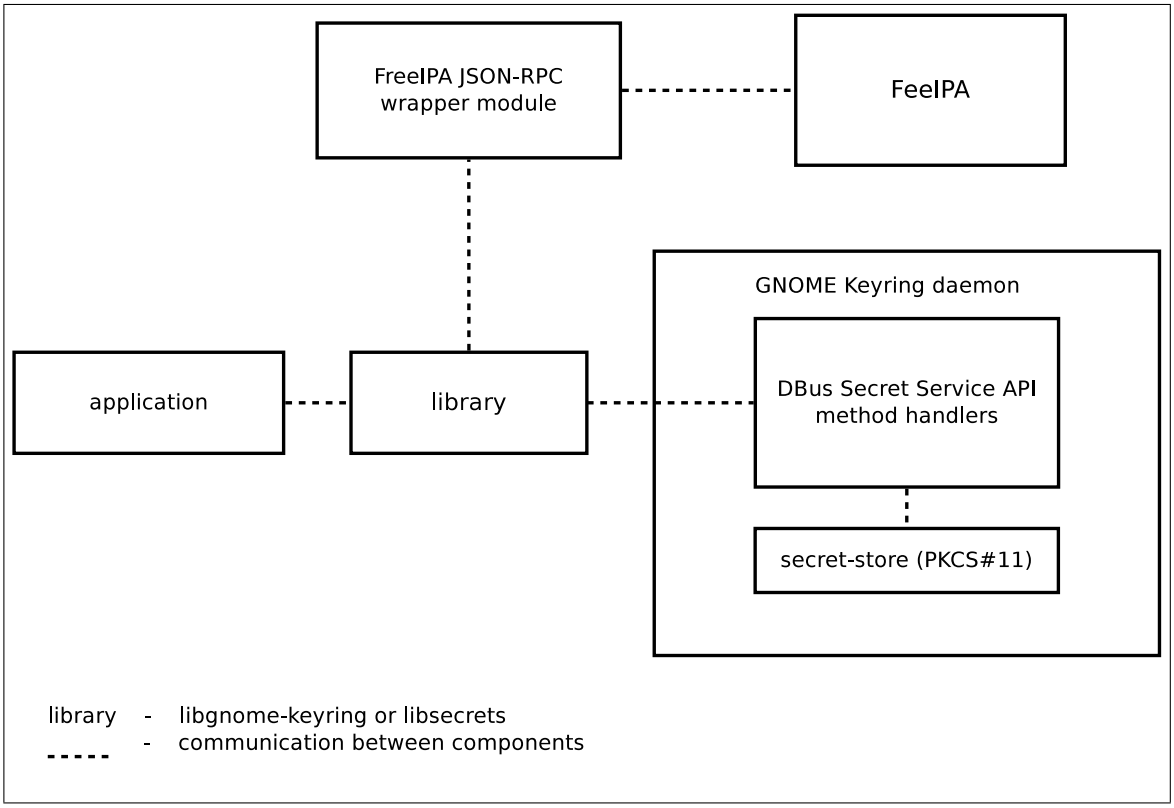


Figure 5.5: Bypassing GNOME Keyring daemon by the library

5.3 Changes Needed on FreeIPA Server

The above approaches are all client side solutions that can use the already existing private vaults in FreeIPA. There are no changes required on the FreeIPA server side other than the KRA component must be installed. However FreeIPA administrators might want to set default settings for enabling/disabling this feature on hosts centrally. More information about this will be in the following section.

5.4 Design for the Prototype

A design demonstrated on Figure 5.3 was chosen for the prototype implementation. The reason for this choice was the hope that this approach will lead to faster development and experimenting with the feature in the initial stages of the development. In comparison to the approach when GNOME Daemon is completely avoided, this approach will cover more applications.

5.5 Functionality of the Prototype

The prototype will support a subset of *libsecret*/*libgnome-keyring* API that is used to manipulate with password. It will be tested on application that will require this functionality to be available:

- storing a password in FreeIPA's vault,
- deleting a password from FreeIPA's vault,
- retrieving a secret/password from FreeIPA's vault,
- all the above operation should be possible to perform on any FreeIPA enrolled host with valid Kerberos TGT with the same results, and
- users should be able to choose if they wish to use the integration or not, administrators can specify defaults for particular users.

  A test application was implemented to test this functionality. More information about how this functionality is tested is provided in Chapter 6.

5.6 Prototype Implementation Details

This section provides implementation details about the most important tasks that need to be done.

5.6.1 Module for Communication with FreeIPA

The first important part that needs to be implemented is a module that will be used for communication with FreeIPA's password vault. The **standard** vault type was chosen as the storage for the secrets in FreeIPA. This type of vault does not protect the secret with any additional secret, so it can operate silently on the background while users are not disturbed with prompts asking for this additional password. The implemented module will use FreeIPA's JSON-RPC API and will need to provide these functions:

- Kerberos authentication against FreeIPA and storing cookie for later use,
- creating a standard vault with specified name,
- checking for existence of vault with specified name,
- archiving secret inside a standard vault,
- retrieving secret from standard vault, and
- deleting a standard vault.

This module is built on top of two private submodules that are not directly exposed to GNOME Keyring code.

The first submodule is responsible for **communication with FreeIPA through network**. It constructs the body of JSON-RPC methods, sends them to the FreeIPA RPC server and returns the response in a buffer. It does not analyze the response and fails only in case of networking issues. This submodule also implements the main function for

Kerberos authentication and stores the cookie retrieved from FreeIPA. This module uses *libcurl* library, which is a library for C often used to implement communication with web applications. The information about what parameters the JSON-RPC methods use was retrieved using the two methods described in 2.5.

The second submodule provides functions that parse the JSON-RPC response from FreeIPA RPC server. These functions check if the remote operation was successful and returns needed data from the response if there are some. In case of retrieve operation it returns base 64 encoded secret from the vault.

The above two modules are used to implement public API for the FreeIPA JSON-RPC wrapper module. The module is called *simple-vault* and his API is documented in the *simple-vault.h* header file.

Every operation that can be performed functions provided by this header must be performed after successful Kerberos authentication which will result in obtaining a cookie file that can be reused for a while before new authentication is required. The authentication function uses libcurl library as every other function in this header. The following subsection will provide some more details on how this is done.

5.6.2 Kerberos Authentication Against FreeIPA with libcurl

It is not possible to use FreeIPA's JSON-RPC API without first authenticating against the FreeIPA server. The result of successful authentication is a session cookie retrieved from FreeIPA that needs to be passed to the subsequent JSON-RPC request. This subsection will show some necessary steps that need to be taken in order to do this with the help of *libcurl* C library. Here only Kerberos authentication is considered, but it is possible to authenticate against FreeIPA using name and password. Note that the code examples shown here do not include checks for errors and use hardcoded value of FreeIPA server hostname in order to make them short.

FreeIPA expects the HTTP referer field to point back to FreeIPA. The following lines can be used to achieve this.

```
headers = curl_slist_append(headers,
                            "referer:https://ipa.example.com/ipa;");
curl_easy_setopt(curl_handle, CURLOPT_HTTPHEADER, headers);
```

The GSS negotiate authentication method will need to be set. The username field in the HTTPS request is not used during this method, but it can not be left empty. So it is possible to put any string in that field. We can see it being done as passing `":"` to the `CURLOPT_USERNAME` in the following two lines.

```
curl_easy_setopt(curl_handle, CURLOPT_HTTPAUTH, CURLAUTH_GSSNEGOTIATE);
curl_easy_setopt(curl_handle,CURLOPT_USERNAME, ":");
```

These two following lines will set path to the cookie jar file. In the prototype implementation, the cookie is stored in the users home dorectory as a hidden file.

```
curl_easy_setopt(curl_handle, CURLOPT_COOKIEFILE, "path_to_cookie.jar");
curl_easy_setopt(curl_handle, CURLOPT_COOKIEJAR, "path_to_cookie.jar");
```

In order for the client to be able to trust the FreeIPA server, it needs to verify it's identity with the CA certificate. The default place to store FreeIPA CA certificate is */etc/ipa/ca.crt*. This file is retrieved during client enrollment.

```
    curl_easy_setopt(curl_handle, CURLOPT_CAPATH, "/etc/ipa/ca.crt");
```

Finally the path to FreeIPA Kerberos login interface on the server is */session/login_kerberos*.

```
    curl_easy_setopt(curl_handle, CURLOPT_URL,
                     https://ipa.example.com/ipa/session/login_kerberos);
}
```

The above are the most notable steps that *libcurl* users need to do in order to get the session cookie. The full implementation of the authentication request can be found in function *ipa_krb_auth_get_cookie()* in the file *src/ipacomm/ipa-jsonrpc.c* on the attached DVD.

5.6.3 Propagating FreeIPA Server Hostname to GNOME Keyring Daemon

The GNOME Keyring daemon needs to know what is the FreeIPA server hostname in order to initiate the communication from the new module. This information is known to the SSSD daemon, that reads it directly from it's configuration or retrieves it from DNS. The SSSD can be modified to pass this information through the PAM responder in an environmental variable, that will be set when the user authenticates successfully with „sss“ PAM plugin. However this mechanism will not be used in the prototype. The problem with passing environmental variables through the PAM stack is that it can only be done during the authentication. The hostname of active FreeIPA server can change at any time, for example if the FreeIPA server becomes unavailable but there are replicas of that server with different hostnames, or due to load balancing the servers can be switched temporarily.

The prototype solution modifies SSSD to store information about the active FreeIPA server in a publicly readable file. SSSD can update this file at any time. Utility function was implemented as part of this prototype to access this file. It is available together with other utility function in vault-utils.h header.

5.6.4 Enabling and Disabling the Integration

The user should be able to enable or disable the integration feature. A special configuration file in the users home directory[1] was added for this purpose. User can specify option GKIPA_ENABLED=TRUE or GKIPA_ENABLED=FALSE to enable or disable integration in this file. If the user does not specify anything in that configuration file or if the configuration file does not exits, the GNOME Keyring will use a utility function to check for existence of flags set by SSSD daemon. These flags are normal files in the `/var/lib/sss/` directory. Their existence or nonexistence has special meaning and GNOME Keyring can act acording to that. The files were used for the same reason as in the hostname propagation issue. If the situation changes (feature is disabled) SSSD can not let users know about this through the PAM stack if the users are already logged in. Wheter or not to set these flags can be ser up in the /etc/sssd/sssd.conf file by using the option store_gnome_keyring_in_ipa. If both the user and the SSSD do not provide any information about wheter the feature should be enabled, the daemon checks if the user is member of *gkusers* group on the FreeIPA server. The FreeIPA administrator can add users to this group if it is desired to have this feature implicitly enabled.

[1] In the current version the file is placed to $HOME/.gkipa.cfg.

5.6.5 Storing the Secrets Locally when the Integration is Enabled

The users may still wish to store the secrets locally in the GNOME Keyrings PKCS#11 *secret-store* storage on the host. This is set up in similar way as enabling the integration feature. First the users home directory is consulted, then SSSD. However FreeIPA admins can not set any defaults for this. This can improve user experience in case when FreeIPA is offline, but on the other hand it makes the secret spread across several hosts, which is one thing the prototype aims to prevent.

5.6.6 Modifying Behavior of DBus Method Handlers in GNOME Keyring Daemon

The prototype solution uses the approach where the DBus module in GNOME Keyring deamon uses functions from the mew communication module to access vault feature in FreeIPA. Even when the integration feature is enabled the collections and items are still created locally and exposed on the DBus, but the secrets are stored in the items only in case when the local storage of passwords was allowed. Otherwise the secrets are stored remotely in FreeIPA's vault.

The Secret Service API uses lookup attributes to look for specific item in a collection. What attributes that are is specified in a schema. For example schema for a person can consists of three lookup attributes: name, surname, date of birth. In order to identify the secrets remotely in appropriate vaults, we need to create vaults that embed these attributes as well as collection name in the vault's name. The vault name could look like this:

```
collection_name:schema_attributes:attribute_values
```

However this string will contain characters that are not allowed in vault's name. To solve this, the vaults name is constructed by encoding this string with custom base 32 encoding. Base 64 could not be used, because it still could contain characters that are not allowed. The base 32 encoding and decoding functions are implemented as part of the *simple-vault* module for the prototype.

5.6.7 Synchronizing the Databases

In order for the integration to work, the prototype still creates empty DBus items for the passwords locally even if the local storage for the actual passwords is not enabled. This helps other applications to detect that the passwords exist, even if they are not available locally but provided as vaults in FreeIPA, because these DBus items provide the interface the passwords.

When the user stores or deletes passwords from session in another host, a special operation needs to be performed to synchronize the host with the FreeIPA database. The operation does the following: Checks for all vaults that serve as containers for GNOME Keyring passwords in the FreeIPA and downloads them. Then it updates the local databe (changes passwords, adds new passwords or deletes passwords if they are no longer present on the server).

This operation is done by a special tool called `ipagksync`. It can be run on demand to update the database at any time or run as a cron job to update the database regularly. However there is a small problem. If the user just recently enabled the integration feature, non of the passwords that are stored locally in his keyring are present on the FreeIPA yet. If the sync operation is executed it will consider the user's passwords old (deleted

from another host) and delete them. In order to prevent this, only passwords that were added while the integration feature was enabled are considered for this ipagksync operation. GNOME Keyring keeps track of these passwords to prevent deletion of passwords that were not delete remotely.

Chapter 6

Testing the Prototype

This chapter provides information about how the implemented prototype was tested and how to run the automated test suite.

6.1 Test Double

A test double was implemented for the purpose of both ad-hoc command line testing as well as in the automated test suite. This test double is a program called `gk-test-ap`. It is a simple command line application that has been written in order to be used for testing in the automated test suite as well as ad-hoc testing in the terminal. It is written using the libgnome-keyring library. The application allows storing, retrieving and deleting different passwords managed by GNOME Keyring.

6.2 Method Used for Testing

The following method was used to test the functionality of the implemented prototype. It uses set of simple scripts that call the `gk-test-app` to do simple simple tasks supported by the application. These are the names of the scripts: A_del_password_1.sh, A_del_password_2.sh, A_get_password_1.sh, A_get_password_2.sh, A_store_password_1.sh, A_store_password_2.sh. The initial letter A means that the script runs the `gk-test-ap inside`. Then the del_password, store_password and get_password values identify the performed operation. The last part is a number 1 or 2. These numbers identify some passwords, so there are two different passwords that can be stored in the GNOME Keyring using these scripts.

Another set of scripts, that is the same as the previous with the difference that the script names begin with B was also implemented. These scripts do not use the test double, but the `ipa` tool to perform the same operations.

The reason for this is that the scripts that are implemented with the `ipa` tool, are doing from the perspective of the test double the same thing as if the test double was executed on another host with the integration feature enabled. If the test double was executed on another host it would result in the same FreeIPA server and client interaction, because both the `ipa` tool as well as the test double do internally use the same JSON-RPC method calls. Both sets of scripts are running under the same user, so it is possible to simulate scenarios where the user stores password on one hosts deletes it on another and then tries to retrieve again from the first host or similar.

This way we can simulate actions of two hosts while needing to setup only one to test against. The host can be the FreeIPA server itself running for example in a virtual machine.

The test are run remotely from another machine and use `sshpass`, a tool similar to `ssh` but allows non interactive logins without the need for having to generate keys for authentication. The script for retrieve operation can be made to expect a success (finding the password) or expecting a failure (expects to not find the requested password). The scripts always informs whether it's expectation was right or wrong.

The tests are run with different configurations. There is a configuration that enables the integration feature, but disables storing the passwords locally, configuration that disables the integration feature completely and configuration that enables both.

Using the above simple scripts it is possible to combine them into different scenarios. A set of these scenarios was implemented and can be run automatically. However, the test suite expects already configured FreeIPA server with patched GNOME Keyring and SSSD. The patches are available in the attached DVD as well as the test scripts.

6.2.1 Test Case Example

Here is an example of one of the test cases.

1. Host A: store password secret123456 into default keyring
2. Host A: try to retrieve the password and expect success
3. Host B: try to retrieve the password and expect success
4. Host A: delete the password
5. Host A: try to retrieve the password and expect failure
6. Host B: try to retrieve the password and expect failure

Of course host B is just simulated with the B_ set of scripts and actually runs on the same host as host A.

6.3 How to Run the Tests

The tests are run using the run-tests.sh script. The README.sh serves as the configuration file for the script. It contains variables with FreeIPA hostname (or IP address) to test against, password for the FreeIPA user and some other variables needed for the tests.

Chapter 7

Conclusion and Future Tasks

This thesis provided necessary information about FreeIPA, GNOME Keyring and related projects in order to understand the motivation behind the effort of integrating these two technologies and in order to understand technical details of possible ways to implement this integration. This chapter summarizes the work being done so far as well as provides information about current efforts in the upstream development of SSSD and how they may be used in the future to integrate GNOME Keyring with FreeIPA.

7.1 Current State of the Prototype

The prototype made as part of this thesis confirmed that the integration of GNOME Keyring with FreeIPA is possible even with very little communication with SSSD. Still, many tasks remain to be done before this prototype is production ready and alternative designs should be further investigated as well, because despite the concept in the current prototype being simple and with the current amount of features also manageable it does not seem to be very extensible and more added features would probably result in closer and closer coupling with SSSD, in which case implementing a new SSSD service would probably be more beneficial in the long term. Also the current SSSD development aims to provide new feature that could be in the future used to simplify GNOME Keyring integration as well. More about it in the next section.

7.2 Secrets Service in SSSD and GNOME Keyring Integration

There is an effort to bring new service to the SSSD called **Secrets Service**. This service will be implemented as a new SSSD responder for storing and managing secrets on behalf of the applications in a local storage or transparently route them to a remote storage like FreeIPA's Vault[6]. This is a parallel effort made by SSSD developers and the author if this thesis was so far not involved in it. However it makes sense to note it here because this service's purpose partially overlaps with this thesis.

The API of this service will be exposed via unix socket and it will be the same or similar API as provided by Custodia, which is a project that aims to provide solution for management of secrets in cloud environments [6].

The purpose of this service and the API that it will provide is very similar to the purpose of GNOME Keyring and it's Secret Service API. And because it is planned to

make it possible to route the secrets to a FreeIPA's Vault, it would implement part of the solution needed for integration of GNOME Keyring with FreeIPA Password Vault using the approach with new SSSD service responder. An approach that was discussed in Section 5.2.

With the service implemented it would be possible to use it's API in the GNOME Keyring daemon to route the secrets to this service. Whether the secrets are routed to FreeIPA or stored locally would be left on the configuration of that service. The information needed to initiate communication with FreeIPA would be kept in SSSD and GNOME Keyring would not need to even know that it is enrolled in FreeIPA domain.

Bibliography

[1] Bokovoy, A.: Talking to FreeIPA API with sessions and JSON-RPC. https://vda.li/en/posts/2015/05/28/talking-to-freeipa-api-with-sessions/.

[2] Foundation, F. S.: GNU C Library manual, Chapter 29: System Databases and Name Service Switch. https://www.gnu.org/software/libc/manual/html_node/index.html.

[3] Kukuk, A. G. M. T.: The Linux-PAM System Administrators' Guide. http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html.

[4] Leupold, S. W. M.: Secret Service API Draft, Chapter 1: Introduction. https://specifications.freedesktop.org/secret-service/ch01.html.

[5] Leupold, S. W. M.: Secret Service API Draft, Chapter 13: Interfaces. http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html.

[6] Sorce, S.: Secrets Service, design document. https://fedorahosted.org/sssd/wiki/DesignDocs/SecretsService.

[7] WWW site: FreeIPA project official website. http://www.freeipa.org/.

[8] WWW site: GNOME Display Manager wiki. https://wiki.gnome.org/Projects/GDM.

[9] WWW site: GNOME Keyring architecture, offcial wiki page. https://wiki.gnome.org/Projects/GnomeKeyring/Architecture.

[10] WWW site: GNOME Keyring official wiki page. https://wiki.gnome.org/Projects/GnomeKeyring.

[11] WWW site: GNOME Keyring wiki, PAM module manual. ttps://wiki.gnome.org/Projects/GnomeKeyring/Pam/Manual.

[12] WWW site: GNOME Project, about page. https://www.gnome.org/about/.

[13] WWW site: GNOME Project, GNOME Technologies page. https://www.gnome.org/technologies/.

[14] WWW site: Intro to Gnome Keyring Keyrings. https://wiki.gnome.org/Projects/GnomeKeyring/KeyringIntro.

[15] WWW site: Manual page for ipa(1) tool. http://linux.die.net/man/1/ipa.

[16] WWW site: OASIS consoritium description. https://www.oasis-open.org/org.

[17] WWW site: OASIS page for PKCS#11.
https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=pkcs11.

[18] WWW site: Offcial site of realmd project.
https://freedesktop.org/software/realmd/.

[19] WWW site: Outdated wiki for FreeIPA XML-RPC API.
http://www.freeipa.org/page/FreeIPAv1:API.

[20] WWW site: Wiki for FreeIPA's Password Vault.
https://www.freeipa.org/page/V4/Password_Vault.

[21] WWW site: Wiki for Password Vault 1.0.
https://www.freeipa.org/page/V4/Password_Vault_1.0.